Grohmann • Eichler

A 68000-es mikroprocesszor

Technika és programozás

DATA BECKER – NOVOTRADE

Grohmann • Eichler

A 68000-es mikroprocesszor

Technika és programozás

DATA BECKER – NOVOTRADE

A könyv eredeti címe: Das Prozessor Buch zum 68 000. Technik & Programmierung (1985)

Fordította: INOTAI LÁSZLÓ

Lektorálta: STANKOVICSNÉ DR. HENCZL ILONA

A kiadásért felel RÉNYI GÁBOR, a NOVOTRADE RT. igazgatója

Felelős szerkesztő: TARR KÁLMÁNNÉ
Műszaki szerkesztő: DÉVÉNYI ERIKA

Szedte a Nyomdaipari Fényszedő Üzem, Budapest
Készült: Nyírségi Nyomda, Nyíregyháza (23 A/5 ív)

ISBN 963 02 4538 8

Hungarian translation © Inotai László

Copyright © 1985 DATA BECKER GmBH – Merowingerstr. 30.4000 Düsseldorf

Minden jog fenntartva. A DATA BECKER cég írásbeli hozzájárulása nélkül tilos a könyvet vagy annak részeit bármilyen eljárással (nyomtatás, fotókópia vagy egyéb technika), elektronikus rendszerek felhasználásával másolni, sokszorosítani, terjeszteni.

FONTOS TUDNIVALÓ

A jelen könyv keretén belül ismertetett kapcsolások, eljárások és programok nem tekinthetők szabadalmi oltalom alá eső ipari termékeknek. Ezek elsősorban amatőr és oktatási célokat szolgálnak. A szerzők rendkívül nagy gondot fordítottak a kapcsolások, műszaki adatok és programok helyességére, a részletek kidolgozása során többszöri ellenőrzést végeztek. Mindez azonban nem zárja ki az esetleges hibalehetőségeket.
Az előforduló hibákért és az ebből adódó következményekért a DATA BECKER cég sem szavatosságot, sem jogi felelősséget nem vállal. Az esetlegesen előforduló hibák közlését a szerzők hálásan fogadják.

TARTALOMJEGYZÉK

ELŐSZÓ 9

BEVEZETÉS 11
A 68000-esig vezető út – a fejlesztés és körülményei 11

1. A 68000-es mikroprocesszor felépítése 13

1.1 Technológia és felépítés 13
1.2 Regiszterszerkezetek és adatszervezés 16
1.3 Üzemi állapotok 23
1.4 Címzésmódok 35
1.5 Utasításkészlet áttekintése 38

2. Jelek és buszok a 68000-esben 41

2.1 A 68000-es jelei és a csatlakozólábak 42
2.2 Aszinkron buszhozzáférés 52
2.3 Szinkron buszhozzáférés 61
2.4 Megszakítások 66
2.5 Buszvonal kérése 69
2.6 Rendszervezérlés 71

3. A család további processzorai 79

3.1 A 68008-as 79
3.2 A 68010-es 83
3.3 A 68020-as 96

4. A 68000-es perifériái 121

4.1 Intelligens perifériaellenőrző egység (IPC) 121
4.2 Buszmegszakító modul (BIM) 122
4.3 Párhuzamos interfész és időzítő (timer) (PI/T) 123
4.4 Közvetlen tárhozzáférés (DMA) 124
4.5 Buszkezelő egység (BAM) 126
4.6 Tárkezelő egység (MMU) 127
4.7 Soros kimeneti-bemeneti egységek 129

4.8 Kombinált soros/párhuzamos kimeneti-bemeneti egységek (MFP) 130
4.9 Lebegőpontos aritmetika társprocesszor (FPP) 130

5. Az utasítások és címzésmódok 135

5.1 Címzésmódok 137
5.2 Kapcsolók (Flags) 155
5.3 Ciklusmód (Loop) 157

6. Az utasításkészlet 159

7. A 68000-es processzor az operációs rendszerekben 329

7.1 A tárkezelő egység (MMU) 332
7.2 Az operációs rendszerek támogatása 336
7.3 A programozási nyelvek támogatása 340
7.4 A 68020-as processzor modultechnikája 343

8. Programozási példák 347

Függelék 355

Felhasznált irodalom 355

Tárgymutató 358

Táblázatok 361

ELŐSZÓ

Ez a könyv a Motorola mikroprocesszor (μP) család szinte valamennyi ismeretanyagát felöleli.
Hogy miért nem mindet?

Egészen egyszerűen azért, mert egy (vagy több) kötetnyi terjedelemben minden részletkérdéssel nem lehet foglalkozni. Azt viszont hisszük, hogy a kezében levő könyv az Ön szemszögéből nézve valamennyi, az Ön számára lényeges témakört átfogóan tartalmazza.

Ez a könyv tehát abban kíván segíteni Önnek, hogy miképpen használja ezt a processzort, és hogy érthetővé tegye a jelenlegi alkalmazási területeit. A könyv magát a hardvert is bemutatja úgy, hogy a meglevő rendszerek bővíthetők, ill. megtarthatók legyenek.

A tapasztalataink szerint annak nincs sok értelme, hogy az egyik kézikönyvből a másikba sok-sok adatot vegyünk át. Sokszor hiányoznak is az éppen ehhez szükséges információk. Könyvünk ezért egyrészt valamennyi, általában szükséges ismeretet közöl, másrészt a használó számára lehetőséget ad arra, hogy a hiányzó adatokat a megfelelő könyvekben megtalálhassa.

Úgy gondoljuk azonban, hogy az ilyen jellegű adatokra csak speciális hardverfeladatok esetén lehet szükség. A rendszer teljes leírása végül is csak azokat érdekelheti igazán, akik kifejezetten ezzel a rendszerrel dolgoznak. Ezért erről – más témakörök javára – lemondunk.

A 68000-es mikroprocesszorral csak jóval a piacra kerülése után kezdtünk el foglalkozni. Pontosan ugyanazzal a problémával találtuk szembe magunkat, mint Ön. Nevezetesen: kell-e vagy akarunk-e dolgozni a 68000-es családdal. Éppen emiatt aztán sok olyan kérdés merült fel, amire – véleményünk szerint – még nem volt megfelelő válasz.

Akkor, amikor ebben a könyvben mikroprocesszorról vagy a 68000-esről beszélünk, ezen mindig a Motorola M68000-esét értjük. Így tehát ha pl. a Motorola 68010-esről beszélünk, akkor ezt csak 68010-esnek fogjuk nevezni.

Az egyes fogalmak gyakran éppen az elektronikában keverednek össze. Ezért – amennyire számunkra ismert – az ésszerű fordításokat használjuk. Az egyes megnevezések kiválasztásánál az általánosan elfogadott szabályokhoz igazodunk. Az olyan megnevezéseket, amelyek tartalma nem érthető, mással helyettesítjük vagy megmagyarázzuk.

A 68000-es felépítéséről szóló fejezetben röviden vázoltuk belső szerkezetének egy részét. Ezzel csak egynéhány elméleti kérdést akartunk gyakorlati megvilágításba helyezni, de semmi esetre sem akartuk a 68000-es teljes „architektúráját" elemezni (ez önmagában egy külön könyv lehetne). Ezért a belső folyamatok vezérlésének egzakt ismertetésétől el is tekintettünk. Csupán annyit mutatunk meg, hogy a 68000-es egy utasítást alapvetően hogyan hajt végre.

Itt köszönjük meg külön Alfred Rozek és Ingo Donasch úrnak az információk beszerzésére vonatkozó segítségét. Ugyanakkor mindazoknak szól a köszönetünk, akik bennünket a korrektúra olvasásánál segítettek.

Köszönjük a Berlini Műszaki Egyetem Szoftver és Elméleti Informatika Intézetének, külön Dr. S. Schindler professzor úrnak, hogy rendelkezésünkre bocsátott egy 68000–UNIX berendezést.

Nyugat-Berlin, 1985. április

Lutz Eichler *Bernd Grohmann*

BEVEZETÉS

A 68000-esig vezető út – a fejlesztés és körülményei

A processzor fejlesztése 1977-ben kezdődött. Ebben az időben a mikroprocesszorokat elsősorban diszkrét logikai áramkörök helyettesítőiként alkalmazták. Ugyanakkor megindult a közepes és a nagy tömegű adatfeldolgozás területén való felhasználásuk is. Olyan alkalmazási területek, mint pl. a szövegfeldolgozás, csak a mikroprocesszorok segítségével voltak feltárhatók.

Ennek során azonban számos probléma merült fel. A monopolhelyzetben levő 8 bites gépek teljesítménye elegendő volt. Az igény viszont sok területen azt diktálta, hogy legyen mód többmunkahelyes rendszerek kialakítására is. Ezt a 8 bites processzorokkal csak a kívánt színvonal alatt lehetett volna megvalósítani. Amint a RAM-elemek egyre olcsóbbá váltak, a professzionális alkalmazók körében egyre kínosabbá vált az a felismerés, hogy a 8 bites processzorokat 64 kbyte-os központi tárnál többre igencsak körülményes lenne alkalmazni.

Másik probléma a processzorok programozhatósága. Korábban jószerint csak az volt az érdekes, hogy a processzor teljesítse feladatát. Arra nem gondoltak, hogy olyan utasításkészletet építsenek fel, amellyel könnyen lehetne modulokból álló programot készíteni.

Az a processzor számított a legjobbnak, amelyik a legtöbb utasítást megértette. A magas szintű nyelvek használatát szinte egyáltalán nem segítették (ehhez maguk a processzorok sem rendelkeztek elegendő teljesítménnyel). Azt, hogy a meglevő tárkapacitás egyre inkább szűkösnek bizonyult, elsősorban nem a programok alacsony hatékonysága vagy áttekinthetetlensége okozta, hanem inkább az, hogy a programot egyáltalán el lehetett-e helyezni a tárban, és hogy azt elfogadható idő alatt le lehetett-e futtatni.

E problémák figyelembevételével kezdődött meg az Austinban (Texas állam) székelő Motorola cégnél a 68000-es család fejlesztése.

A 68000-es (pontosabban M68000) név az előző típus, az MC68000-as nevéből származik. Ez a processzor néhány jellemzőjét (pl. az ugróutasításokat) tekintve annak idején a keresztapa szerepét töltötte be.

Ezen túlmenően néhány, más processzornál megkedvelt tulajdonságot – javított formában – be kívántak építeni a 68000-esbe. Így átvették az IBM és az INTEL központi egységeinek számos funkcióját. Ezek utasításkészletei szolgáltak példaként a 68000-es fejlesztői számára.

A 68000-es keresztelőjén a Digital Equipment (DEC) PDP–11 (ill. LST–11) mikroprocesszora is jelen volt. A fejlesztés során a szoftver egyre jelentősebb szerepet kapott.

A 68000-esnek olyan tulajdonságai vannak, amelyek a moduláris programozást és a magas szintű programnyelvek használatát megkönnyítik. A különböző alkalmazási rendszerek a 68000-esen egyszerűen valósíthatók meg.

A 68000-es felépítése családszerkezetű. Ez azt jelenti, hogy a teljesítményt illetően léteznek különböző változatok, ezek azonban a programok vonatkozásában egymással teljes mértékben kompatíbilisek.

A család ma létező processzorai a teljes utasításkészlet (ami egészében még nincs is meghatározva) csak egy részével rendelkeznek. Az újabb processzorok alkalmazását így a szoftverek felfelé való kompatibilitása megkönnyíti. A füzérek (string-ek) feldolgozására és a lebegőpontos aritmetikára a család első processzorainál így nincs lehetőség.

A processzorban viszont bőven van hely arra, hogy további funkciókat és utasításokat lehessen beépíteni.

Egy mikroprocesszor típus elterjedésénél igen fontos szempont, hogy az hányféle perifériális egységgel tud dolgozni. Mivel egy processzor és a hozzá tartozó perifériális egységek egyidejű fejlesztése aligha lehetséges, a Motorola a 68000-est úgy tervezte meg, hogy a 6800-as sorozat perifériális egységei közvetlenül alkalmazhatók legyenek. Így a 68000-es a piaci bevezetéskor a perifériális egységek (szinte) teljes választékával rendelkezett már.

Aligha vásárol valaki egy olyan alkatrészt, amit csak egy gyártó cég árul. A vásárló számára fontos, hogy azt a bizonyos alkatrészt máshol is megkaphassa. Ez jótékonyan hat a versenyre, és a szűk kapacitásokat is megszünteti. A második vonalban (Second Sources) levő gyártók között azonban vannak különbségek. A felhasználó számára az a lényeges, hogy az egyes alkatrészek egymás között teljes mértékben kompatíbilisek legyenek. A 68000-est a Motorola (MC68000) mellett még a Signetics/Phillips/Valvo (SC68000), a Hitachi (HD68000), a Thomson CSF (EF68000) és a Rockwell (R68000) is gyártja. A gyártóknak már a puszta felsorolása is mutatja, hogy mennyire értékeli a piac a 68000-es mikroprocesszort.

1. A 68000-ES MIKROPROCESSZOR FELÉPÍTÉSE

1.1 Technológia és felépítés

Egy ennyire összetett mikroprocesszor előállításához a félvezetők akkori gyártástechnológiáját is tovább kellett fejleszteni. A 68000-es mikroprocesszorhoz kb. 68000 tranzisztor-, ill. 13 000 kapuműveletre van szükség. Az akkori NMOS-technológia nem tette lehetővé az integráció olyan nagy sűrűségét, ami a chip méretét gazdaságilag megfelelő határokon belül tarthatta volna. Így jött létre a HMOS- (High density MOS) technológia. Ez az NMOS-hoz képest kb. kétszeres áramkörsűrűséget tett lehetővé. Emellett a sebesség- és a veszteségi teljesítmény értékeket négyszeresére javította. A 68000-es kb. 1,5 Watt teljesítményt igényel. Ha NMOS-technológiával készülne, akkor azonos működési sebesség mellett a veszteségi teljesítmény 6 W lenne.

A központi vezérlőegységet (CPU) alapvetően kétféleképpen lehet felépíteni. A lényeg a vezérlőmű felépítésén és az utasítások dekódolásán van. A vezérlőmű tisztán diszkrét módon is megtervezhető, ami azt jelenti, hogy csak kombinációs logikai áramköröket – kapu áramköröket – tartalmaz. Ezt a technikát random logikának nevezik. A random alatt itt szabad választást kell érteni, és a logika csak a műveleti kritériumokra épül.

A másik lehetőséget mikroprogramozásnak nevezik. Itt a műveletek vezérlését egy ROM veszi át. Ez már szinte úgy is felfogható mint „egy mikroprocesszor egy másik mikroprocesszoron belül”.

A következő táblázat a kétféle eljárás közötti különbségeket mutatja be:

	Szabadon választott (random) logika	**Mikroprogramozás**
Felépítés	szabálytalan	szabályos
Áttekinthetőség	alig van	van
Változtathatóság	csak nagy munkával változtatható	csak a mikroprogramot kell változtatni
Bővíthetőség	csak nagy munkával lehetséges	viszonylag egyszerű (ha van szabad ROM-terület a mikroprogram számára)
Fejlesztési idő	igen hosszú	lényegesen rövidebb

	Szabadon választott (random) logika	**Mikroprogramozás**
Helyigény	igen nagy	a mikroprogram-technikától függően lehet viszonylag kicsi vagy nagy
Komplexitás	nagy	közepes
A feldolgozás sebessége	nagy	a mikroprogram-technikától függően lehet viszonylag kicsi vagy nagy

Minthogy a 68000-es kifejlesztésénél az egyik legfontosabb követelmény éppen a bővíthetőség volt, gyakorlatilag csak a processzor mikroprogramozása jöhetett szóba. A 68000-est csak a mikroprogramozás alkalmazásával lehetett gazdaságosan kifejleszteni. A lábkiosztás vonatkozásában gyakorlatilag kompatíbilis vezérlőegységeket, mint pl. a 68010-est, ily módon viszonylag rövid idő alatt meg lehetett valósítani.

A mikroprogramot tartalmazó ROM-nak kell a végrehajtó egységet vezérelnie. Ehhez az utasításdekódoló egy kezdőcímet ad meg a ROM számára. A végrehajtó egység erről a ROM-területről kapja meg a vezérléséhez szükséges egyes biteket. A 68000-es végrehajtó egységének kb. 180 vezérlési pontja van.

Egy mikroprogramot tartalmazó ROM-nak alapvetően két formája van, egy úgynevezett VÍZSZINTES és egy úgynevezett FÜGGŐLEGES mikrokód:

– A vízszintes mikrokód esetén a végrehajtó egység számára gyakorlatilag mindegyik vezérlési ponton egy bit áll rendelkezésre. A ROM-nak igen nagynak kell lennie, viszont nagymértékben egyszerűsíti az egyidejű (párhuzamos) feldolgozást, hiszen a vezérlő információ gyakorlatilag dekódolva áll rendelkezésre. Így a 68000-es valamely utasításának végrehajtásához csak néhány mikroutasításra (ezek a mikrokód ROM sorok) van szükség. A feldolgozási sebesség ily módon igen nagy, mindez viszont nagy ROM-területet igényel.

– A függőleges mikrokód esetén a kód szélessége kicsi. Ezért ehhez komplex logikára van szükség, hogy az a végrehajtó egységnek szóló parancsokat dekódolhassa. Egy makroutasítás (ami a 68000-es valamely utasításának felel meg) függőleges mikrokódban több mikroutasítás együttesét jelenti. Mivel a mikroutasítások feldolgozásának sebessége technológiai korlátok miatt tetszőleges mértékben nem növelhető, a központi végrehajtó egység (CPU) sebessége szükségszerűen csökken. A függőleges mikrokód előnye az, hogy kis területű ROM kell hozzá.

Elvileg a mikroprogramot tartalmazó ROM-ot a CPU-n kívül is el lehet helyezni: Ilyen rendszerű pl. a DEC (Digital Equipment) az LSI–11 típusú gyártmánya.

Ebben az esetben külső vezérlőműről beszélünk (ellentétben a belső vezérlőművel). Mivel azonban az egység tokozásán levő csatlakozólábak száma korlátozott, vízszintes mikrokód gyakorlatilag nem használható. Ez csökkenti a processzor feldolgozási sebességét. A csatlakozólábak multiplex használata ugyan szélesebb mikrokód ROM tárterületet tesz lehetővé, de ez is csökkenti a feldolgozási sebességet. Ezért a belső vezérlőművet kell előnyben részesíteni. Az LSI–11 esetében ez műszaki okok miatt azonban még nem volt lehetséges.

A 68000-esnél vegyítették a kétféle mikrokódtechnikát. Létrehozták a kétlépcsős mikrokódot, amelyet „hibrid vertikális-horizontális struktúrának" is neveznek. A 68000-esben két ROM létezik. Ezek a mikrokód ROM és nanokód ROM névre „hallgatnak".

A nanokód ROM nagy szélességű (70 bit). Így mindössze 3 nanoutasítás (a nanokód ROM sorai) szükséges ahhoz, hogy a végrehajtó egység valamennyi vezérlési pontjához hozzá lehessen férni. Mivel az utasítássor nem mindegyikéhez kell valamennyi vezérlési pont, gyakran elegendő kevesebb nanoszó is. Emellett a nanoszavak átmenetileg tárolhatók is. A 68000-esben kb. 280, egyenként 70 bit hosszúságú nanoszó van. A nanokód ROM vezérléséhez (címzéséhez) tehát elegendő 9 bit.

A mikrokód ROM-ot az utasítást dekódoló egység hívja meg. Mindegyik makroutasítás (felhasználói utasítás) több mikrokódsorból áll. A mikrokód mindegyik sora most azonban (a tisztán függőleges mikrokóddal ellentétben) egy nanokódsorra mutat. Mivel a nanoszavak igen szélesek, a makrokód több része párhuzamosan feldolgozható, így minden egyes mikroutasítás részsorozatokra (szekvenciákra) ágazhat el. Ezeknek csak egyszer kell meglenniük a ROM-ban, és így a chipen a vezérlőmű kisebb helyet foglal el. A 68000-es utasításkészletének beépítéséhez kb. 640 mikrokódsorra van szükség. Minden sorhoz 9 bit kell, hogy ezek egyértelműen utalhassanak egy nanokódsorra. Ehhez jön még egy extra bit, ami az elágazásokhoz szükséges. A mikrokód ROM tehát összesen kb. 640 sorból, soronként 10 bitből áll. A vezérlőmű teljes tárterülete tehát kb. 26 kbit. Ha a gyártók csak mikrokódot alkalmaztak volna, akkor kb. 45 kbit tárterületű ROM-ra lett volna szükség.

A vezérlőmű sebességének további növelése érdekében még egy újabb módszert alkalmaztak. Akkorra, amikor egy utasítás végrehajtása megkezdődik, a műveleti kód (opcode) és az operandus (ill. egy további utasítás, amennyiben előzőleg egy egyszavas utasításról volt szó) behívása már megtörtént. Ezt „prefetch techniká"-nak (= előzetes behívásos technika) nevezik. A gyakorlatban ez azt jelenti, hogy egy utasítás feldolgozásának (tehát a mikro- és a nanokód ROM-hoz való hozzáférésnek) lehetőleg más „tevékenységekkel" átfedésben kell megtörténnie. Így jobban kihasználható a központi vezérlőegység, és nagyobb lesz az utasítások végrehajtási sebessége.

Ezen kívül a regiszterműveletek párhuzamosan futhatnak, mert a címekhez és az adatokhoz külön-külön számlálóegység (ALU = aritmetikai-logikai egység) áll rendelkezésre. Így pl. egy regiszter-összeadási művelet és a programszámláló növelése egyidejűleg történhet. Ez is gyorsítja az adatforgalmat.

Mivel a 68000-es utasításkészlete a ROM-okba be van építve, elvileg lehetséges lenne a felhasználók igényei szerinti, maszkolt programozás. A felhasználónak így lehetősége lenne arra, hogy saját utasításkészletet állítson össze. Ez azonban (egy kivételtől eltekintve) még nem szerepel a Motorola kínálatában, mert több érv szól ellene. Vevőorientált CPU-k esetén nem lennének egységes assemblerek és disassemblerek, emulátorok és analizátorok. A 68000-es ára a kisebb darabszámok miatt magasabb lenne. A szoftvereket is külön-külön kellene elkészíteni az egyes maszkváltozatokhoz. Felhasználói szempontból a Motorola e döntését üdvözölni kell.

1.2 Regiszterszerkezetek és adatszervezés

A 68000-es két különböző üzemmódban dolgozik. Ezeket „supervisor"-módnak és „user"-módnak nevezik (felügyeleti és felhasználói mód). Csak a supervisor-módban használható valamennyi utasítás. Ez a struktúra teszi lehetővé a többmunkahelyes rendszerek biztonságos működtetését. Ehhez már csak egy MMU-ra (Memory-Management-Unit) = tárkezelő egység) van szükség. Ez az egyike azon lényeges előnyöknek, amellyel a 68000-es az IBM–PC-ben levő 8086-ossal (iAPX 86) szemben rendelkezik. Erre a témára az üzemállapotok ismertetésénél és a perifériaegységekkel foglalkozó fejezetben térünk ki részletesebben.

A felhasználó számára a 68000-esben nyolc adatregiszter áll rendelkezésre. Mindegyik adatregiszter 32 bit szélességű. Ezért a 68000-est gyakran 32 bites processzorként emlegetik. Mivel azonban az adatbusza 16 bit szélességű, a processzort is 16 bites processzornak kell tekinteni. Az adatregiszterek jelölése D0...D7.

Emellett van hét címregiszter és egy programszámláló. E regiszterek szélessége is 32 bit. Ez tulajdonképpen 4 gigabyte-nyi tárterületet jelent. Mivel a 68000-es címbusza azonban csak 24 bit szélességű, a felhasználó számára „csak" 16 megabyte áll rendelkezésre. A címregiszterek jelölése A0...A6, a programszámlálóé PC (Program-Counter).

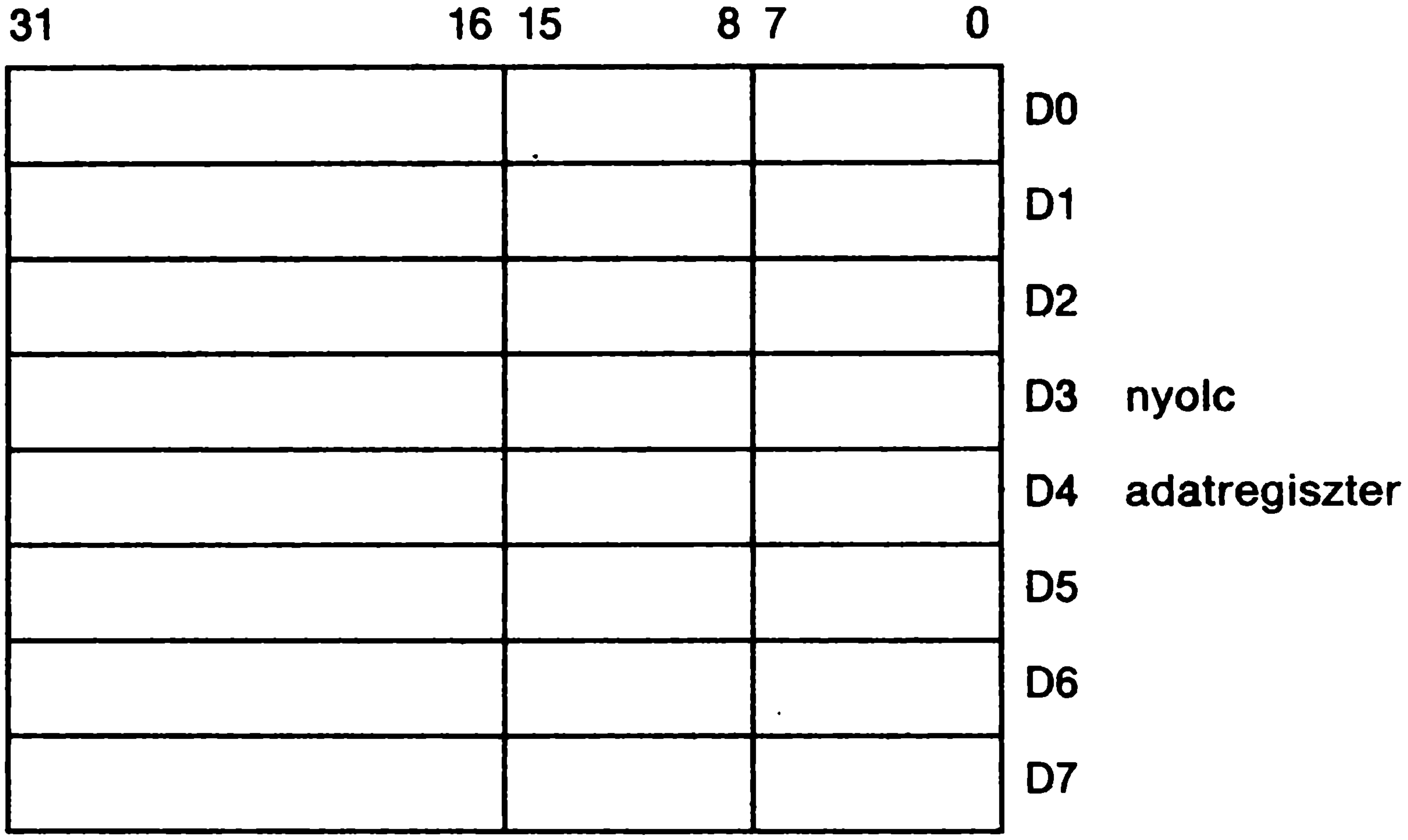
A processzorban két veremmutató (stackpointer) regiszter van, az egyik a user-, a másik a supervisor-mód (felhasználói és felügyeleti üzemmód) számára. A CPU üzemmódjától függően aktív az egyik veremmutató. A veremmutatók jelölése A7 és A7'. Már a jelölésből is látszik, hogy az aktív veremmutató mint a 7-es jelölésű címregiszter is használható. Még az olyan utasításokkal is, amelyek általában egy címregiszterhez, és nem implicit módon a veremmutatóhoz szólnak, csak éppen aktív veremmutató érhető el. Kivételt képez ez alól a „MOVE USP" utasítás. Azáltal, hogy két veremmutató regiszter van, igen egyszerű módon lehet külön a user és külön a supervisor számára egy-egy vermet kezelni.

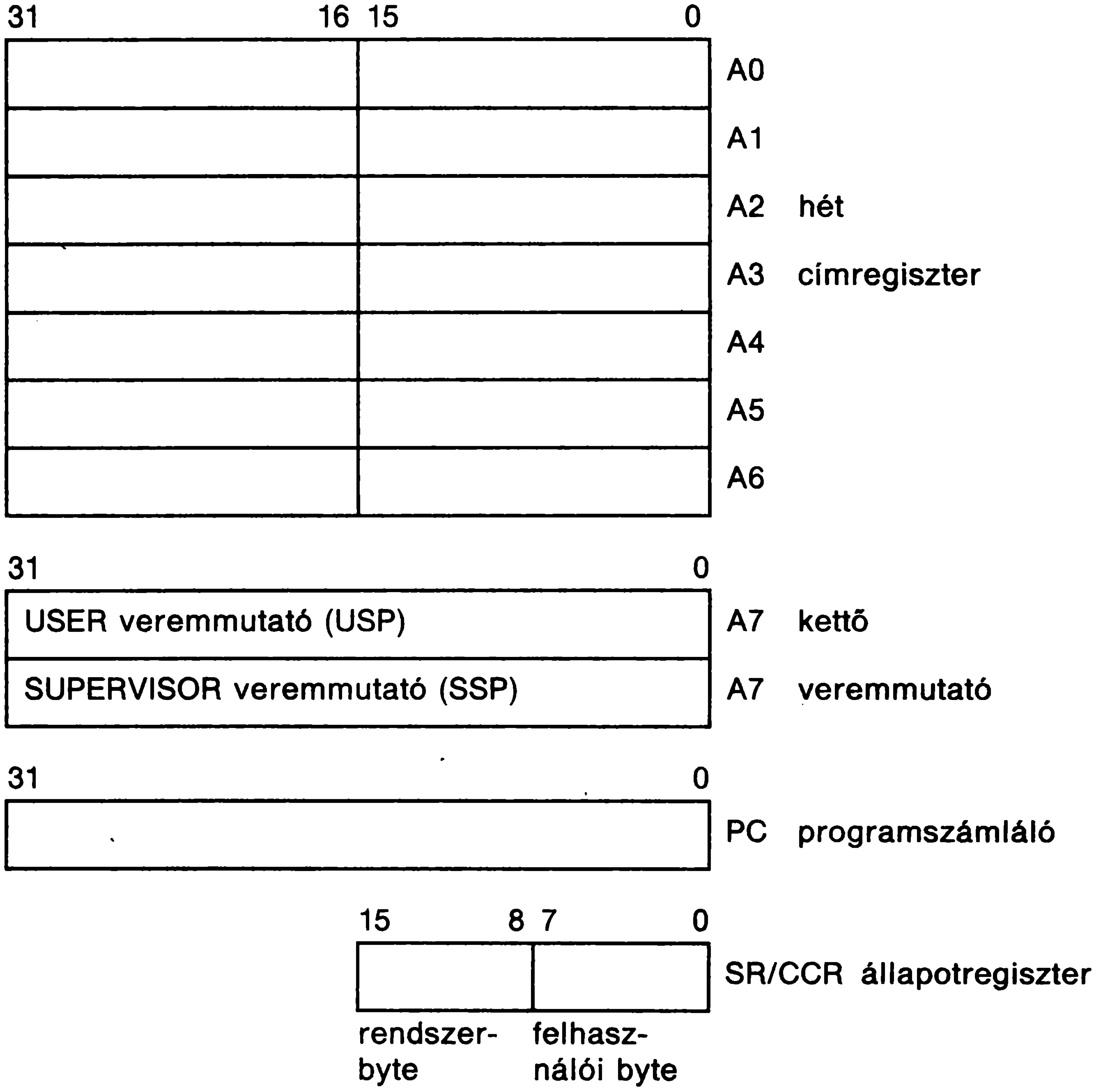
Valamennyi adat- és címregiszter indexregiszterként is használható. Az univerzálisan használható regiszterek nagy számának köszönhető, hogy a compilerek (fordítók) és interpreterek (értelmezők) könnyen illeszthetők a 68000-eshez. Ezáltal válik a 68000-es nagy teljesítményű és a magas szintű programnyelvet is megértő processzorrá. Ugyanakkor utasításkészlete annyira áttekinthető, hogy assembler nyelven is jól programozható.

A 68000-es programszámlálója (PC) 32 bit szélességű. Ebből most „csak" 24 bit jut a címbuszra. A többi 8 bit a későbbi bővítésekre van fenntartva (így válik a címtár 4 gigabyte-ossá).

Végül, de nem utolsósorban, van egy állapotregiszter (SR), amely 16 bit szélességű. Ennek felosztása: felhasználói byte (userbyte) – 0...7 bit – és rendszerbyte (systembyte) – 8...15 bit –. Felhasználói (user-) módban csak a felhasználói byte-hoz lehet hozzáférni. Így tehát csak a supervisor változtathatja meg a CPU üzemmódját (más értelmetlen is lenne). A felhasználói byte tartalmazza azokat a kapcsolókat (flags), amelyeket a feltételes ugróutasítások (branch on...) kérdeznek le.

A 68000-es regiszterei:




31
16 15
0
A0
A1
A2 hét
A3 címregiszter
A4
A5
A6
31
0
USER veremmutató (USP)
SUPERVISOR veremmutató (SSP)
A7 kettő
A7 veremmutató
31
0
PC programszámláló
15
8 7
0
SR/CCR állapotregiszter
rendszer-byte
felhasználói byte

Az operandusok formátuma:

Az operandus formátumát vagy implicit módon az utasítás tartalmazza, vagy explicit módon az utasításban előre meg kell adni.

A következő formátumú operandusok definiáltak:

1 kettős-szó (longword)	megfelel 32 bitnek
1 szó (word)	megfelel 16 bitnek
1 byte	megfelel 8 bitnek

BCD (binárisan kódolt decimális) operandusok feldolgozására is van lehetőség. A feldolgozásuk „tömörítve" történik, ami azt jelenti, hogy két-két BCD-szám egy byte-ba összefogva kerül feldolgozásra. Ezen kívül vannak még bitmanipulációs utasítások.

Az operandusok standard formátuma a szó, mert a 68000-es adatbusza 16 bites.

A regiszterek szervezése

Az adatregisztereknél az operandusok valamennyi formátuma megengedett. A byte hosszúságú operandusok a legalsó 8 biten, a szó hosszúságú operandusok a legalsó 16 biten helyezkednek el. A kettős-szó hosszúságú operandusok mind a 32 bitet elfoglalják.

Ha egy adatregisztert forrás- vagy céloperandusként használunk, és az operandus hosszúsága nem éri el a 32 bitet, akkor a regiszternek mindig csak az a része változik, amihez „szólunk”. A regiszter többi részére nincs szükség, és tartalma sem változik.

A címregisztereknél és a két veremmutatónál csak szó és kettős-szó hosszúságú operandusok használhatók. A címregiszterek és a veremmutatók byte- és bitadatokkal nem dolgoznak. A veremmutatók mindig a legutolsó érvényes adatra mutatnak, és úgy nőnek, ahogy a címek csökkennek.

Ha egy címregiszter forrásoperandusként van megadva, akkor az operandus választott formátumától függően vagy a teljes regiszter, vagy csak az alsó szó kerül felhasználásra. Ha egy címregisztert céloperandusként használunk, akkor ezzel – az operandus választott formátumától függetlenül – a teljes regisztert befolyásoljuk. Ha a formátum szó típusú, akkor valamennyi operandus előjelesen 32 bitre bővül.

Állapotregiszter

Az állapotregiszter egy felhasználói és egy rendszerbyte-ból áll. Ezek felépítése:

Felhasználói byte

0. bit: átvitelkapcsoló (Carry – C)
1. bit: túlcsorduláskapcsoló (Overflow – V)
2. bit: zérókapcsoló (Zero – Z)
3. bit: negatívkapcsoló (N)
4. bit: bővítőkapcsoló (extension – X)
5. bit: nem használt
6. bit: nem használt
7. bit: nem használt

8...10. bit:	megszakítási maszk (Interrupt-Mask I0, I1, I2)
11. bit:	nem használt
12. bit:	nem használt
13. bit:	supervisor állapot
14. bit:	nem használt
15. bit:	nyomkövetési mód (trace)

A következőkben ismertetjük az egyes kapcsolók jelentését:

Átvitelkapcsoló:

Az átvitelkapcsoló értéke mindig 1, ha egy aritmetikai művelet során a céloperandus legmagasabb bitjén átvitel történt.

Túlcsorduláskapcsoló:

A túlcsorduláskapcsoló azt jelzi a felhasználó számára, hogy egy aritmetikai művelet elvégzése során túlléptünk a számtartományon. Ez pl. abban az esetben következik be, ha két pozitív szám összeadásakor az eredmény már nem fér el a regiszterben, ha pedig egy negatív szám, akkor kettes komplemensben van ábrázolva. A túlcsorduláskapcsoló osztási műveleteknél azt jelzi, hogy a hányados 16 bitnél nagyobb lenne, vagy hogy az osztandó túl nagy.

Zérókapcsoló:

A zérókapcsoló értéke 1, ha egy összehasonlítás azt mutatja, hogy a két operandus azonos, ill. ha a művelet eredménye nulla.

Negatívkapcsoló:

A negatívkapcsoló értéke akkor lesz 1, ha egy művelet során a kapott eredmény legmagasabb helyiértékű bitjének az értéke 1, tehát az eredmény egy negatív szám kettes komplemensben.

Bővítőkapcsoló:

A bővítőkapcsoló ugyanúgy viselkedik, mint az átvitelkapcsoló. Az értékét viszont nem változtatja meg annyi utasítás, mint amennyi az átvitelkapcsolóét. Pl. az összeadásokra és kivonásokra ugyanúgy reagál, mint az átvitelkapcsoló, az elforgatási utasításokra viszont bizonyos körülmények között nem. A bővítőkapcsoló használatára pontos felvilágosítást az utasításjegyzék ad. A bővítőkapcsoló arra is használható, hogy az átvitelkapcsoló tartalmát több „segédművelet" után is megőrizze. A bővítőkapcsoló a 68000-es egyik specialitása.

Az állapotregiszter felhasználói byte-jának többi – nem használt – bitjei mindig nullák, még akkor is, ha azokba más értéket akarnánk írni. Ugyanez érvényes az állapotregiszter rendszerbyte-jának nem használt bitjeire is.

Következzék most a rendszerbyte bitjeinek a leírása:

Megszakítási maszk (Interrup-Mask):

A 68000-esnek hét megszakítási szintje van (ezek 1-től 7-ig vannak számozva). Egy megszakítás csak akkor engedélyezett, ha a megszakítási maszk értéke kisebb, mint a megszakítás prioritási foka. A 7-es megszakítási szint azonban soha nem zárható le. Ez az úgynevezett NMI (Non-Maskable-Interrupt), a nem maszkolható megszakítás. A megszakítási maszk változtatásával a megszakításokat tehát le lehet tiltani és fel lehet szabadítani.

Supervisor állapot:

Ezzel a bittel kapcsolható át a processzor a user- (felhasználói) és a supervisor-állapot között. Ha a bit értéke 0, akkor a processzor user-állapotban, ha értéke 1, akkor supervisor-állapotban van. Az átkapcsolás teszi lehetővé, hogy a többmunkahelyes rendszerek „ne állhassanak fejre".

Nyomkövetési mód (Trace):

Ha a bit értéke 1, akkor a 68000-es nyomkövetési módban dolgozik.
A 68000-es ilyenkor minden egyes utasítás feldolgozása után egy ún. kizárásfeldolgozási üzemmódba (Exception) kapcsol. Így a 68000-esen a lépésenkénti üzemmód csak megfelelő szoftver segítségével lehetséges.

Az állapotregiszter rendszerbyte-jának felhasználását a 68000-es üzemi állapotairól szóló fejezetben még részletesebben ismertetjük.

Az adatok tárbeli szervezése

Bár a 68000-es egy 16 bites processzor, mégis úgy dolgozik, mint egy byte-beosztású gép. Ez azt jelenti, hogy mindegyik szó két byte-ra van osztva, és e byte-ok mindegyikének saját címe van. A 68000-essel a tárban levő szó mindkét byte-ját természetesen egyidejűleg lehet elérni, hiszen az adatbusz 16 bit szélességű. A tárban tehát mindegyik byte egy-egy címet foglal el, egy szó tehát két címen helyezkedik el. Egy szó (az 'n' címen levő szó) magasabb helyiértékű byte-ja az alacsonyabb címen (az n címen), az alacsonyabb helyiértékű byte pedig a magasabb címen (az n+1 címen) helyezkedik el.

A leírtakat az alábbi vázlat szemlélteti:

15 8	7 0	
000000 byte	000001 byte	000000 szó
000002 byte	000003 byte	000002 szó
00004 byte	000005 byte	000004 szó
⋮	⋮	
FFFFFC byte	FFFFFD byte	FFFFFC szó
FFFFFE byte	FFFFFF byte	FFFFFE szó
páros cím	páratlan cím	

Az adatok tárbeli szervezése
(a szavak szervezése)

15 12 11 8	7 4 3 0	
0. byte	1. byte	n szó
2. byte	3. byte	n+2 szó

Egész számú adatok (byte)

15 12 11 8 7 4 3 0	
0. szó	n szó
1. szó	n+2 szó
2. szó	n+4 szó

Egész számú adatok (szó)

15 12 11 8 7 4 3 0		
kettős-szó felső fele	0. kettős szó	n szó
kettős-szó alsó fele		n+2 szó
kettős-szó felső fele	1. kettős szó	n+4 szó
kettős-szó alsó fele		n+6 szó

Egész számú adatok (kettős-szó)

15 12	11 8	7 4	3 0	
BCD 7	BCD 6	BCD 5	BCD 4	n szó
BCD 3	BCD 2	BCD 1	BCD 0	n+2 szó

BCD 7: legmagasabb helyiértékű szám
BCD 0: legalacsonyabb helyiértékű szám
Decimális adatok (BCD-kódolt)

A 68000-es felépítéséből adódóan a tár elérését bizonyos szabályok korlátozzák:

1) A szavakhoz vagy a kettős-szavakhoz hozzáférés csak PÁROS számú címekről történhet.
2) A műveleti kódoknak (utasításoknak) ezért PÁROS számon kell állniuk.
3) Byte-formátumban a tár PÁROS és PÁRATLAN számú címről is elérhető.

A szabályok figyelmen kívül hagyása a normál feldolgozást megszakítja, és a processzorban a kizárási feldolgozást (Exception) indítja el.

1.3 Üzemi állapotok

A 68000-esnek alapvetően három üzemi állapota van:

1) Normál feldolgozás
2) Kizárási feldolgozás (Exception)
3) Leállási állapot

A leállási állapot részletes tárgyalásától eltekintünk.

A normál feldolgozás: amint erre a megnevezés is utal – az utasítások normál végrehajtását jelenti. Ennek az állapotnak egyik különleges esete a CPU megállított állapota. Ezt az állapotot a STOP utasítás váltja ki. Ebben az állapotban a tárolóhoz a továbbiakban már nem lehet hozzáférni.

A kizárási állapotot a megszakítások, a TRAP utasítások (szoftveroldali megszakítások), a nyomkövetési üzemmód (race) vagy más kizárási feltétel váltja ki. A kizárási állapot implementálása teszi lehetővé, hogy a processzort hibák vagy előre nem látható szituációk esetén is bizonyos reakciókra lehessen késztetni. Ez különösen a többmunkahelyes rendszereknél és általában a nagyobb rendszereknél fontos.

A leállási állapot olyan, igen súlyos hibák esetén áll be, amikor abból kell kiindulni, hogy a rendszer már nem működőképes. A processzort ebből az állapotból csak egy külső jellel (RESET) lehet kimozdítani. A leállási állapotot pl. egy korábbi buszhiba következtében beálló kizárási feldolgozás közben fellépő buszhiba (kettős buszhiba) idézheti elő. A leállási állapot és a megállított állapot azonban nem ugyanaz!

Az állapotregiszter rendszerbyte-jában levő supervisor-bit értékétől függően a 68000-es vagy supervisor állapotban (ha ennek a bitnek az értéke 1), vagy felhasználói (user-) állapotban dolgozik. A mindenkori privilegizált állapot határozza meg, hogy mely műveletek a megengedettek. A user-állapotban néhány utasítás tiltott. Ha ezeket mégis használnánk, akkor ez kizárási feldolgozáshoz (Exception) vezet. A supervisor-állapotban mindig az A7¯ veremmutató regiszter, a user-állapotban az A7 regiszter jut szerephez. Annak nincs jelentősége, hogy eközben a veremmutató regisztert egy utasítás implicite használja-e (pl. PEA), vagy hogy az A7 regiszter egy utasításban célként vagy forrásként explicite meg van-e adva.

A 68000-es ezen kívül három funkciókimeneten (FC0...FC2) jelzi azt, hogy az éppen megkezdett hozzáférés melyik fajtája van a buszon. Ennek során az egyes hozzáférési módokat a három kimenet az alábbiak szerint kódolja:

FC2	**FC1**	**FC0**	**A ciklus típusa**
LOW	LOW	LOW	nincs meghatározva
LOW	LOW	HIGH	user állapot, adatok
LOW	HIGH	LOW	user állapot, program
LOW	HIGH	HIGH	nincs meghatározva
HIGH	LOW	LOW	nincs meghatározva
HIGH	LOW	HIGH	supervisor állapot, adatok
HIGH	HIGH	LOW	supervisor állapot, program
HIGH	HIGH	HIGH	a megszakítás nyugtázása

E kijelzésen keresztül ismeri fel az MMU (Memory Management Unit), hogy a hozzáférési módok melyikéről van szó. Így bizonyos címtartományok a felhasználó elől elzárhatók, ugyanakkor van lehetőség a virtuális tár meghatározására. Ez olyan mechanizmusokat tesz lehetővé, amelyek egy számítógépes rendszer biztonságát megnövelik. Elkülöníthetők olyan tartományok, amelyekre a felhasználóknak nincs szükségük, ill. amelyeket nem szabad megváltoztatniuk. A dolog lényege, hogy a programok többsége user-állapotban (felhasználói állapotban) futtatható legyen, és a supervisor-állapotot gyakorlatilag csak az operációs rendszernek kelljen használnia.

Valamennyi kizárási feldolgozás (Exception) supervisor állapotban történik. Ha egy kizárási folyamatot kell elindítani, akkor a processzor az éppen meglevő állapotszót tárolja a verembe, és a supervisor bitet 1-re kapcsolja. A kizárások feldolgozása során a buszciklusok supervisor-ciklusként kerülnek kijelzésre. A processzor supervisor állapotában valamennyi utasítás használható.

A supervisor-állapotban van arra lehetőség, hogy a felhasználói (user) veremmutató regiszterbe (A7, USP) írni, illetve onnan olvasni lehessen. Normál esetben a supervisor-állapotban nincs lehetőség az A7 regiszter (USP) elérésére, mert csak az $A7^-$ regiszter elérhető. A USP-t, a „MOVE USP" utasítással érhetjük el. Így lehetőség van arra, hogy segédprogramokkal és rendszerrutinokkal felvilágosítást kapjunk a felhasználói programok teljes menetéről. A MOVE USP utasítás csak a supervisor-állapotban megengedett. Ez az első ránézésre értelmetlennek tűnhet, hiszen a felhasználó a user-állapotban minden további nélkül hozzáférhet az A7 regiszterhez. Ez az utasítás valószínűleg azért kapott valamiféle előjogot, mert csak a supervisor-állapotban van értelme. A fejlesztők minden bizonnyal azt várták ettől, hogy az ilyen jellegű hibákat idejében „ki lehessen szűrni".

A kizárások feldolgozása (Exception)

A kizárások feldolgozása kívülről és belülről is kiváltható. Belső kiváltást okozhat pl. címzéshiba (páratlan számú címen levő szót akarunk elérni), a nullával való osztás, közvetlen utasítások (TRAP-utasítások) vagy a nyomkövetési mód (TRACE). Kívülről a kizárásokat a megszakítások, buszhibák vagy a RESET jel válthatja ki.

A 68000-es számos kizárást ismer. Nem utolsósorban ez az egyik erőssége. A kizárások teszik lehetővé, hogy a processzor egyértelműen reagáljon a kizárási állapotokra és a hibákra. Ebben a tekintetben a 68000-es számos mikroszámítógépet és a legtöbb mikroprocesszort felülmúlja.

Az egyes kizárási esetek számozva vannak, és a processzor a bekövetkezett esetnek megfelelő kizárási vektort veszi elő. Ez a vektor egy 32 bites címre mutat, amelynek a tárolása ugyanúgy történik, mint bármely más címé. A legalsó 1024 byte (vagy 512 szó) áll táblázatként a 256 vektor rendelkezésére.

A kizárások feldolgozásának megkezdésekor tárolásra kerül az állapotregiszter tartalma. Ezután a supervisor-bit az 1 értékre áll. A TRACE-bit visszaállítódik, hogy emiatt a kizárási állapot ne indulhasson újra. Ha a kizárás oka egy letiltás volt (ami csak akkor lehetséges, ha a megszakítás prioritása nagyobb, mint amit az állapotregiszterben a megszakítási maszk állása megadott), akkor ezen kívül a megszakítási maszk az állapotregiszterben az új értékre áll be. A visszaugrás címe és az állapotregiszter korábbi tartalma a supervisor-verembe kerül.

A processzor a vektor számát kétféle módon kaphatja meg. Vagy saját maga állítja elő belsőleg (pl. busz- vagy címhibák, de autovektor-megszakítások esetén is), vagy pedig az úgynevezett „nem autovektor megszakítás" (Non-Autovektor-Interrupt) esetén a buszról (közvetlen vagy közvetett úton arról a készülékről, amely a megszakítást kiváltotta). A 68000-es a vektorszámot négygyel megszorozza (a vektorszám bitjeinek kétszeri balra tolásával). Az így kapott számot címként használja. Erről a címről egy kettős-szót tölt be, és visz át a programszámlálóba. Ezután kezdi meg annak az utasításnak a végrehajtását, amelyre a programszámláló (új) értéke mutat, és így tér rá a kizárások feldolgozására.

A RESET az egyetlen olyan kizárás, amelynek két vektora van. Itt az egyik vektor a supervisor-veremmutató (SSP) kezdőcímére, a másik vektor a programszámláló kezdőcímére mutat. RESET esetén egymás után töltődik be a supervisor-veremmutató (A7¯) és a programszámláló.

A RESET vektor betöltésekor buszhozzáférés történik a supervisor programterületre.

A következő táblázat tartalmazza a kizárási vektorokat:

A vektor száma (hexa)	**A vektor címe (hexa)**	**A vektor jelentése**
00	000	RESET (a supervisor-veremmutató kezdőcíme)
01	004	RESET (a programszámláló kezdőcíme)
02	008	buszhiba
03	00C	címhiba
04	010	illegális utasítás
05	014	osztás nullával
06	018	CHK utasítás (CHecK = vizsgálat)
07	01C	TRAPV utasítás (szoftveroldali megszakítás)
08	020	privilégium megsértése
09	024	TRACE nyomkövetés
0A	028	nem implementált utasítás (műveleti kód: $Axxx)
0B	02C	nem implementált utasítás (műveleti kód: $Fxxx)

A vektor száma (hexa)	A vektor címe (hexa)	A vektor jelentése
CC	030	foglalt
0D	034	foglalt
0E	038	foglalt
0F	03C	nem inicializált megszakítás
10–17	040–05C	foglalt
18	060	hibás megszakítás
19	064	megszakítás autovektor (1-es szint)
1A	068	megszakítás autovektor (2-es szint)
1B	06C	megszakítás autovektor (3-as szint)
1C	070	megszakítás autovektor (4-es szint)
1D	074	megszakítás autovektor (5-ös szint)
1E	078	megszakítás autovektor (6-os szint)
1F	07C	megszakítás autovektor (7-es szint)
20–2F	080–0BC	TRAP utasításvektor
30–3F	0C0–0FC	foglalt
40–FF	100–3FC	„nem autovektorok" (felhasználói megszakítások), négy-négy byte-onként

A foglalt vektorokat a használó lehetőleg ne használja, 'nem autovektorként'. Ellenkező esetben e család más processzoraival kapcsolatban inkompatibilitások adódhatnak elő.

A kizárások (Exceptions) ismertetése:

RESET (A7$^-$/SSP)

Hardver-RESET esetén ennek a címnek a tartalma a supervisor-veremmutató regiszterbe (SSP) töltődik be.

RESET (PC)

Hardver-RESET esetén ennek a címnek a tartalma a programszámlálóba töltődik be.

Buszhiba

Buszhiba esetén (ennek a kijelzése az "nBERR" csatlakozólábon történik) a processzor ezen a vektoron levő címre ugrik. Buszhibát egy külső készülék jelez. Buszhibát válthat ki pl. az is, ha egy RAM-ban paritáshiba keletkezik, ha egy „tiltott" tárterülethez fordulunk, vagy ha egy olyan címet kívánunk elérni, amely nincs benne a hardverben.

Címhiba

Ezt a kizárást az váltja ki, ha egy szó vagy kettős-szó műveletnél páratlan számú címet akarunk elérni. Ha címhiba lép fel, akkor a CPU ugyan végrehajtja a normál buszhozzáférési műveletet, de elnyomja az nLDS és az nUDS jeleket, és nem várakozik az nDTACK jelre. [A rövidítések magyarázatával a könyv 2. fejezete foglalkozik – Fordító megjegyzése.]

Illegális utasítás

Ezt a kizárást az váltja ki, ha egy olyan, nem implemetált utasítás érkezik, aminek a szintaxisa nem $Axxx vagy $Fxxx. Ugyanez a kizárási állapot lép fel akkor, ha egy nem implementált címzési módot használunk. A felhasználónak azonban – hogy a kompatibilitások hiányát el lehessen kerülni – csak a $4AFC utasítást kell kiadnia, mert a Motorola a másik két műveleti kódot részben az emulátorok töréspontjainak (break-point) elhelyezésére használja.

Nullával való osztás

Ez a kizárási állapot akkor áll elő, ha egy osztás során a nevező nulla.

CHK utasítás (CHecK = vizsgálat)

A CHK utasítással azt lehet megvizsgálni, hogy egy adatregiszter értéke bizonyos határokon belül van-e. Ez a kizárási állapot akkor lép fel, ha az adatregiszter értéke nem esik a definiált határok közé.

TRAPV utasítás

Ez a kizárási állapot akkor következik be, ha kiadtuk a TRAPV utasítást, és az állapotregiszter felhasználói (user-) byte-jában (CCR = Condition-Code-Register) a túlcsordulásbit értéke 1.

Privilégium megsértése

Ez a kizárási állapot akkor következik be, ha a CPU felhasználói (user-) üzemmódban van és egy privilegizált utasítást adunk ki.

Trace

Ha az állapotregiszter Trace-bitjének az értéke 1, akkor minden egyes utasítás után az ezzel a vektorral kijelölt programrészbe ugrunk. Így egy program lépésenkénti kidolgozásához nincs szükség hardverkiegészítésre.

Nem implementált utasítás (műveleti kód: $Axxx)

Ezt a kizárási állapotot az váltja ki, ha egy olyan utasítás kiadásának végrehajtását kíséreljük meg, amelynek a négy legnagyobb helyiértékű bitje "$A".

Nem implementált utasítás (műveleti kód: $Fxxx)

Ezt a kizárási állapotot az váltja ki, ha egy olyan utasítás kiadásának a végrehajtását kíséreljük meg, amelynek a négy legnagyobb helyiértékű bitje "$F".

Nem inicializált megszakítás

Ha azoknak a külső egységeknek a megszakítási vektorregiszterei, amelyek nem alkalmasak autovektor-megszakításokra (Non-Autovektorinterrupt), nincsenek inicializálva, és egy megszakítás lép fel, akkor ezek az egységek egy „nem autovektor szám"-ként $0F-t adnak ki, és kiváltják ezt a megszakítási állapotot. Így ismerhetők fel a nemkívánatos és a nem inicializált megszakítások.

Hamis megszakítások

Ez a kizárási állapot akkor következik be, amikor egy megszakítás visszaigazolása során buszhiba lép fel. Egy valódi megszakítás esetén a megszakítás logikai áramköre vagy az a külső egység, amelyik a megszakítást előidézte, nyugtázza a 68000-es megszakítási jelentését, és szükség esetén a kizárási vektorszámot az adatbuszra adja. Így lehet pl. azokat a zavaró impulzusokat, amelyek egy megszakítást váltanak ki, a „valódi" megszakításoktól megkülönböztetni.

Autovektor megszakítások

A processzor mindazt a beérkező megszakítást nyugtázza, aminek az állapotregiszter által korábban megadotthoz képest elsőbbsége (prioritása) van. A processzor számára a külső logikai nyugtázó áramkörnek kell jeleznie azt, hogy egy autovektor vagy egy nem autovektor megszakításról (Non-Autovektor-Interrupt) van-e szó. Ha autovektor megszakítás történt, akkor a 68000-es „megnézi" az ennek megfelelő autovektor megszakítást (lásd a táblázatot).

TRAP utasítások (TRAP #0...#15)

A TRAP utasítás segítségével a felhasználói programokkal a célnak megfelelően beléphetünk a supervisor-módban futó programokba. Így pl. teljesíthetők az operációs rendszer igényei, vagy megvalósíthatók a szoftveroldali megszakítások. Amikor a CPU egy TRAP utasítást dekódol, akkor kizárási állapotba vált át, és betölti a megfelelő vektort. A következő táblázat a vektorok és a TRAP utasítások egymáshoz rendelését mutatja:

A vektor száma (hexa)	A vektor címe (hexa)	Utasítás
20	80	TRAP #0
21	84	TRAP #1
22	88	TRAP #2
23	8C	TRAP #3
24	90	TRAP #4
25	94	TRAP #5
26	98	TRAP #6
26	9C	TRAP #7
28	A0	TRAP #8
29	A4	TRAP #9
2A	A8	TRAP #10
2B	AC	TRAP #11
2C	B0	TRAP #12
2D	B4	TRAP #13
2E	B8	TRAP #14
2F	BC	TRAP #15

Nem autovektor megszakítások (Non-Autovektor-Interrupts)
Amint már írtuk, egy megszakítást követően valamely külső egység kizárási vektorokat is megadhat. Ehhez a 192 vektorszám egyikét kell megadni, hogy az inkompatibilitásokat és más zavarokat elkerüljük. A 68000-es legtöbb perifériális egysége ki tud váltani nem autovektor megszakításokat.

A kizárások feldolgozása lehetővé teszi azt, hogy a processzor a hibákat nemcsak megtalálja, hanem azokra céltudatosan reagálni is tud. Még egy boszorkánykonyhában sem igen lehet annyi receptet kitalálni, mint amennyit e processzor valamennyi kizárási állapotára fel lehetne használni. A lehetőségek a felhasználói program számára szóló jelentés kiadásától az egész rendszer megállításán keresztül a gépet kezelő ember megszólításáig terjednek. Ha egy bizonyos kizárási állapotra a megfelelő reakció még nincs is definiálva, az operációs rendszernek egy kizárási állapot bekövetkeztét legkevesebb jelentenie kell. Így a rendszerhibák időben kideríthetők.

A kizárási állapot vektorait értelemszerűen a RAM-ban kell tartani. Így van arra lehetőség, hogy szükség esetén a változtatásokat végre lehessen hajtani. Ahhoz viszont, hogy a teljes rendszer zavarait megakadályozzuk, egy MMU-val meg kell tudni akadályozni, hogy a felhasználó (aki a CPU felhasználói állapotban van) hozzáférhessen a vektortáblához.

A teljes rendszer hidegindításakor a RESET-vektoroknak és a BOOT-programnak természetesen a ROM-ban kell lenniük. A vektorokat aztán át kell másolni egy „párhuzamos" RAM-ba, a ROM-ot ki kell kapcsolni, és helyette a legalsó 1024 byte-on levő RAM-ot be kell kapcsolni.

A kizárási állapot prioritása

Természetesen az is előfordulhat, hogy a véletlenek szerencsétlen egybeesése következtében egyidejűleg, vagy majdnem egyidejűleg több kizárási állapot lép fel. A CPU-nak ilyenkor is a helyzet urának kell maradnia. Ezért a kizárási állapotokat három, különböző prioritási csoportba sorolták. Az egyes kizárásoktól függően ezek feldolgozása különböző időpontokban kezdődik meg. Ez azért lényeges, mert ezáltal a prioritásokat könnyen meg lehet változtatni.

Ha két kizárási állapot nagyjából egy időben következik be, akkor először a nagyobb prioritású kerül feldolgozásra. Ezután a másik kizárási állapot feldolgozása került sorra, és a processzor ezt követően visszakapcsol a normál utasítások végrehajtására. A „nagyjából egyidejűség" alatt azt kell érteni, hogy a processzornak abban az időpontban, amelyikben azt ellenőrzi, hogy fel kell-e dolgoznia egy kizárási állapotot, egyszerre több követelménynek kell eleget tennie.

Magán a 0-ás és az 1-es jelű csoporton belül is az egyes kizárási állapotok (csökkenő sorrendben) különböző prioritásúak. A 2-es csoportba sorolt kizárási állapotok egyidejűleg nem léphetnek fel, mivel ezeket felhasználói utasítások váltják ki.

A 0-ás csoportnak a legnagyobb, a 2-es csoportnak a legkisebb a prioritása:

Csoport	**Kizárás**	**A feldolgozás megkezdése**
0	RESET	két ütemcikluson belül
0	buszhiba	
0	címhiba	
1	TRACE-üzem	a következő utasítás előtt
1	megszakítás	
1	illegális utasítás	egy buszciklus végén
1	nem implementált	
1	privilégium megsértése	
2	TRAP #n, TRAPV, CHK	az ezt kiváltó utasítást követően
2	osztás nullával	

Ha egy buszhiba, egy címhiba vagy egy visszakapcsolás feldolgozása során buszhiba lép fel, akkor a processzor felfüggeszti a működését, és lekapcsolódik a buszról. Így a tár tartalmát a processzor nem teheti tönkre. A működésében megállt processzort a RESET bemeneten keresztül lehet újraindítani.

A veremtartalom kizárások következtében bekövetkezett változása

Amint már említettük, a processzor a visszatéréshez szükséges információkat a supervisor-verembe helyezi el. Közönséges esetben csak a programszámláló állását és az állapotregiszter tartalmát kell tárolni. Egy kizárásfeldolgozási rutin indításakor természetesen a többi regiszter tartalmát is be lehet tölteni a verembe.

Busz- és címhibák esetén a processzor több információt helyez el a verembe. Ez azért szükséges, mert a programszámláló állása (a CPU „prefetching”-je miatt) a hiba fellépésének időpontjáról semmilyen biztos felvilágosítást sem tud adni, és a kizárások feldolgozása az éppen futó buszciklus közben kezdődik meg. Ezen kívül olyan kizárásfeldolgozási rutint is lehetővé kell tenni, ami elemzi a fellépett hibát.

A kiegészítő információk alatt az utasításregiszter tartalmát (tehát az éppen feldolgozás alatt álló utasítás első szavát), azt a címet, amelyen a hiba fellépett és egy kiegészítő szót értünk.

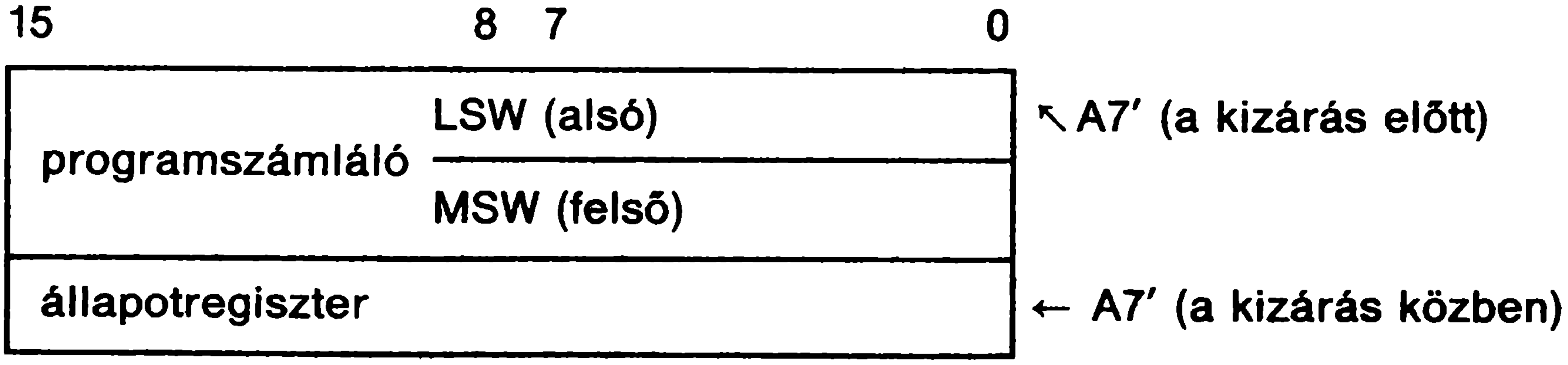
A kiegészítő szó az alábbiak szerint épül fel:

0...2 bit: ezek a bitek a funkciókód-vezetékeknek egy hiba fellépése során bekövetkezett állapotát tartalmazzák.

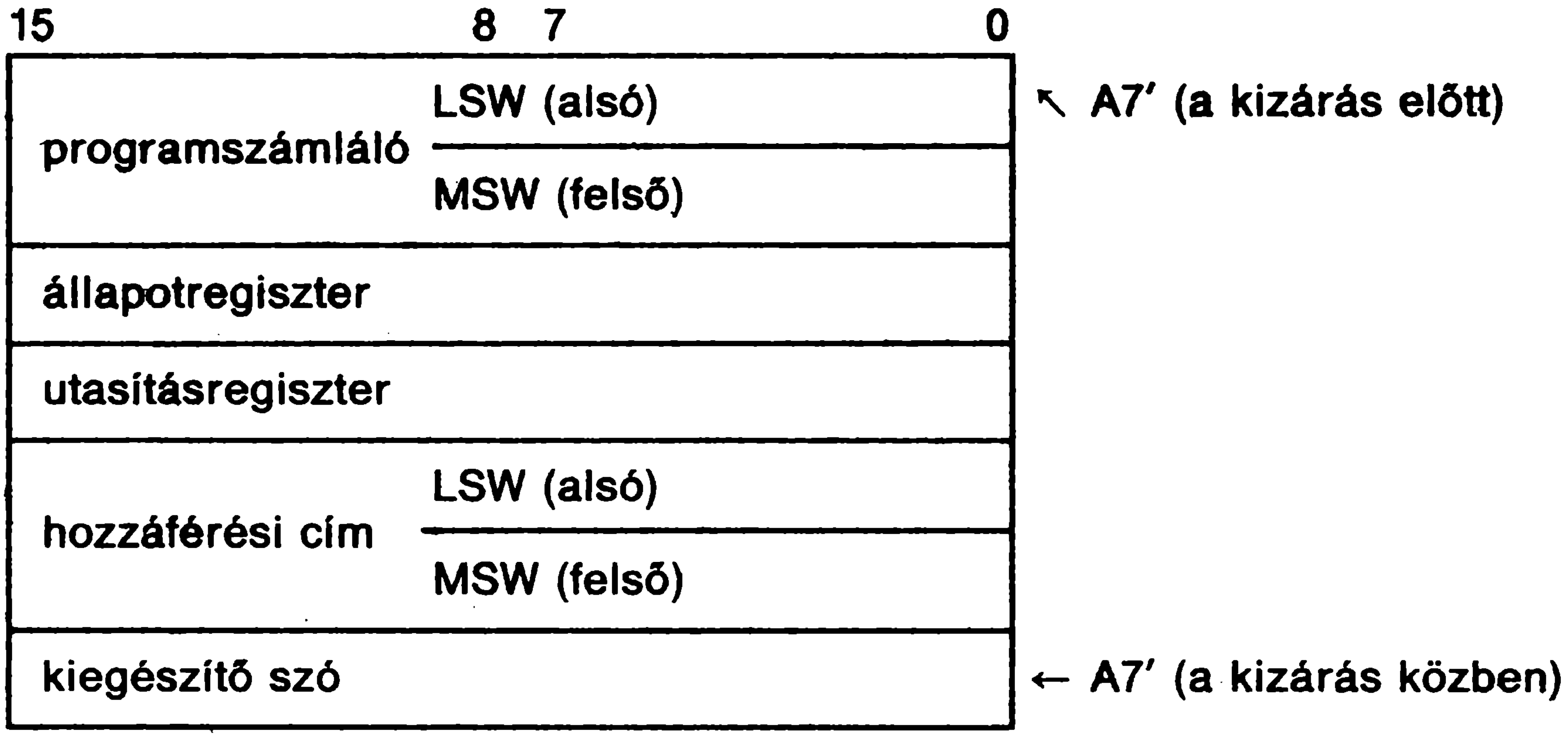
3. bit: ez a bit nulla, ha megszakításkor egy utasítás vagy egy, a 2-es csoportba tartozó kizárási állapot feldolgozása történt, és 1 az értéke, ha egy, a 0-ás csoportba tartozó kizárási állapot vagy egy megszakítás került feldolgozásra.

4. bit: ez a bit akkor nulla, ha előzőleg írási, és akkor 1, ha előzőleg egy olvasási ciklus volt.

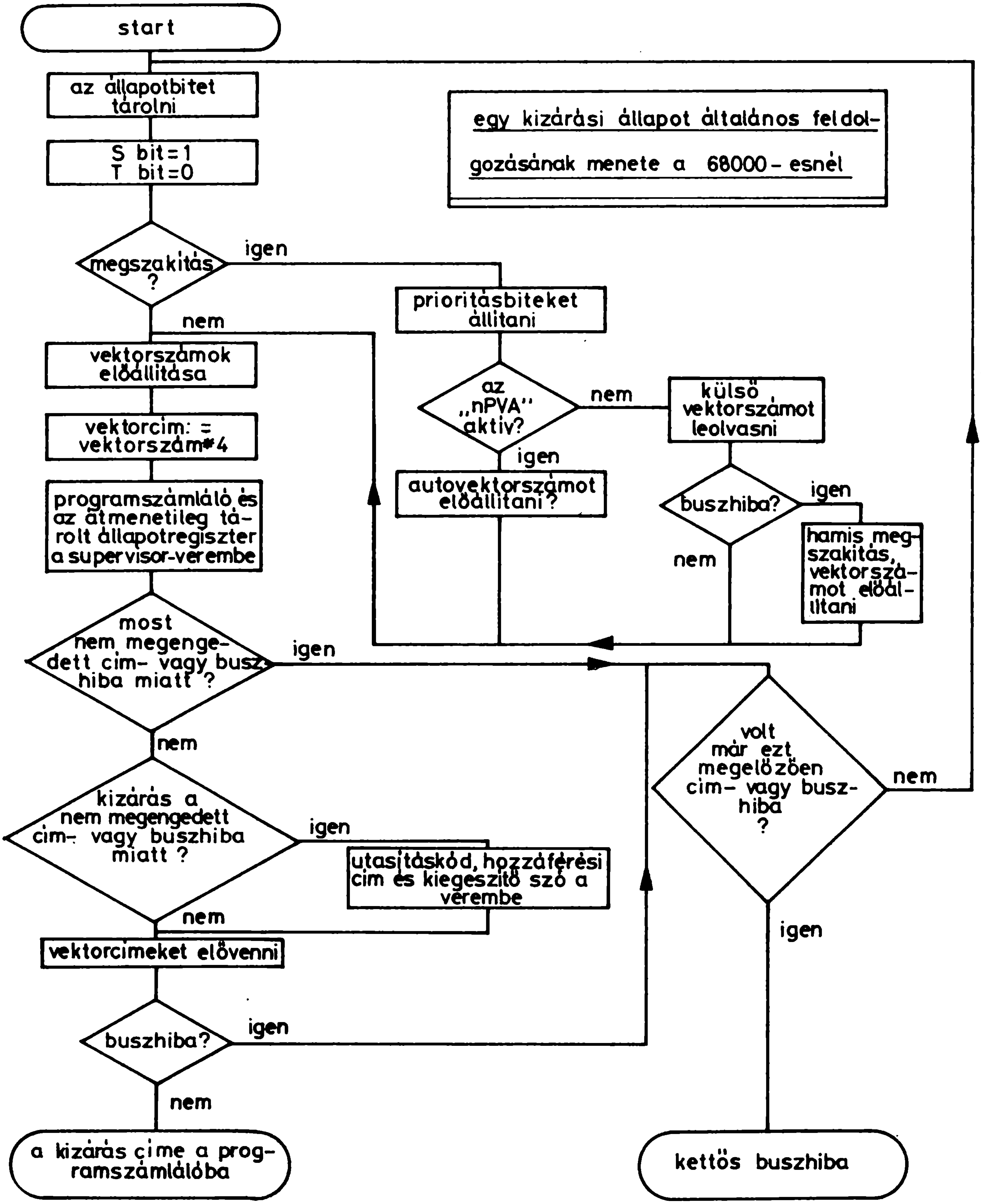
5...15. bit: nem használt.



A supervisor-veremben levő információk (az 1-es és a 2-es csoportba tartozó kizárások esetén)


15
8
7
0
programszámláló
LSW (alsó)
MSW (felső)
állapotregiszter
utasításregiszter
hozzáférési cím
LSW (alsó)
MSW (felső)
kiegészítő szó
↖ A7′ (a kizárás előtt)
← A7′ (a kizárás közben)

A supervisor-veremben levő információk (a 0-ás csoporba tartozó kizárások esetén)


start
az állapotbitet tárolni
S bit=1
T bit=0
megszakítás ?
igen
nem
egy kizárási állapot általános feldolgozásának menete a 68000-esnél
prioritásbiteket állítani
vektorszámok előállítása
az „nPVA" aktiv?
nem
külső vektorszámot leolvasni
igen
vektorcim: = vektorszám*4
autovektorszámot előállitani ?
buszhiba?
igen
nem
programszámláló és az átmenetileg tárolt állapotregiszter a supervisor-verembe
hamis megszakitás, vektorszámot előállitani
most nem megengedett cim- vagy buszhiba miatt ?
igen
nem
volt már ezt megelőzően cim- vagy buszhiba ?
nem
kizárás a nem megengedett cim- vagy buszhiba miatt ?
igen
utasitáskód, hozzáférési cim és kiegészitő szó a verembe
nem
igen
vektorcimeket elővenni
buszhiba?
igen
nem
a kizárás cime a programszámlálóba
kettős buszhiba

A supervisor-verem szervezése a kizárások feldolgozása során

1.4 Címzésmódok

A 68000-es utasításai a következő két részből állnak:
1) A végrehajtandó művelet fajtája
2) Az operandus(ok) címe

Cím alatt itt azonban nemcsak a tároló címét kell érteni, hanem ez természetesen egy regiszter is lehet.

Az utasítások az operandusok címét három különböző módon határozhatják meg:

1) Regiszter specifikáció: a regiszter számát az utasításban adjuk meg.
2) Effektív cím: ahhoz, hogy a címet megkapjuk, különböző címzésmódokat kell használni. A kiválasztás az utasításban levő hat bittel történik (ezt a hat bitet elfoglaló területet nevezzük effektív címnek).
3) Implicit hivatkozás: az operandust (egy regisztert) az utasítás implicite már tartalmazza.

A 68000-esnek 14 címzésmódja van, amelyek az operandusoknak az előzőekben megadott valamelyik címzésmódját használják. Ezt a 14 címzésmódot hat fő csoportba lehet osztani:

Egy regiszter közvetlen címzése:
Az utasításban közvetlenül megadjuk azt a regisztert, amelyik az operandust tartalmazza.

Egy regiszter közvetett címzése:
Az utasításban egy olyan regisztert adunk meg, amelyik az operandus abszolút címét tartalmazza.

Adatok abszolút címzése:
Az utasításhoz a kiegészítő szavakban a tárban levő operandus abszolút címét adjuk meg.

Relatív címzés a programszámlálóhoz:
A programszámláló számára egy relatív távolságot adunk meg. Ez azt jelenti, hogy a programszámlálóhoz előjelhelyesen egy szót vagy egy kettős-szót adunk hozzá. Az eredmény az operandus címe. Ezek a címzésmódok teszik lehetővé olyan programok írását, amelyek a rendszer tetszőleges címétől kezdődően futtathatók.

Adatok közvetlen címzése:

Egy utasítás kibővítésében (egy vagy két szó) van közvetlen az operandus.

Implicit címzés:

Az operandust az utasítás implicit módon tartalmazza. A veremműveletek tartalmazzák pl. implicit módon azt a veremmutatót, amely az operandus címére mutat.

A 14 címzési módból 13 az utasításban úgynevezett effektív címet képez. Ez az effektív cím az (első) utasításszóban kétszer három bit területet foglal el. Az egyes címzési módoktól függően az utasításszót még további szavak követhetik, amelyek az operandust közelebbről határozzák meg.

Az effektív című utasításoknál az utasításszó a következőképpen épül fel:

0...2. bit:	Ezek a bitek tartalmazzák a regisztermezőt.
3...5. bit:	Ezek a bitek tartalmazzák az üzemmódmezőt. (A 0...5 bit képezi az effektív címet.)
6...11. bit:	Ezek a bitek vagy egy további effektív címet, vagy az utasítás egy másik részét képezik.
12...15. bit:	Ezek a bitek tartalmazzák az utasítás típusát.

A következő táblázat azokat a címzési módokat mutatja be, amelyek egy effektív cím által kerültek kiválasztásra.

Ebben a következő rövidítéseket használtuk:

ARI: közvetett címregiszter
An: egy címregiszter száma (3 bit, tehát 0...7)
Dn: egy adatregiszter száma (3 bit, tehát 0...7)

Effektív cím		**Címzési mód**
mód	**Regiszter**	
000	Dn	adatregiszter közvetlen
001	An	címregiszter közvetlen
010	An	címregiszter közvetett (ARI)
011	An	ARI utólagos növeléssel (posztinkrementálás)
100	An	ARI előzetes csökkentéssel (predekrementálás)
101	An	ARI címkülönbséggel
110	An	ARI címkülönbséggel és indexszel
111	000	abszolút, rövid

Effektív cím Mód	 Regiszter	Címzési mód
111	001	abszolút, hosszú
111	010	programszámláló relatív, címkülönbséggel
111	011	programszámláló relatív, címkülönbséggel
111	011	programszámláló relatív, címkülönbséggel és index-szel
111	100	adatok közvetlen (immediate)

Az implicit címzési módnál effektív címre nincs szükség.

Címregiszter közvetlen címzése, utólagos növeléssel (posztinkrementálással):
Az operandus címe a megadott címregiszterben van. A művelet után a címregiszter az operandus hosszától függően 1-gyel, 2-vel vagy 4-gyel növekszik. Ha a veremmutatóról van szó, akkor a cím minimum 2-vel növekszik, hogy a veremmutató értéke páros maradjon. Ezzel a címzési móddal további vermek építhetők fel.

Címregiszter közvetlen címzése, előzetes csökkentéssel (predekrementálással):
A megadott regiszter 1-gyel, 2-vel vagy 4-gyel csökken. Amennyiben a címregiszter esetén a veremmutatóról van szó, akkor ez 2-vel vagy 4-gyel csökken, hogy páros számú maradjon. Ez megakadályozza a címhibákból adódó kizárásokat. Azt a címet érjük el, ami a kivonás után a megadott címregiszterben található.

Címregiszter közvetlen címzése, címkülönbséggel:
Ennél a címzési módnál a megadott címregiszter tartalmához egy további szó adódik hozzá. Szükség van tehát egy bővítő szóra. Az operandus címe a regiszter tartalmának és a címkülönbség 16 bites, előjelhelyes értékének az összege lesz.

Címregiszter közvetlen címzése, címkülönbséggel és indexszel:
Ez a címzési mód megegyezik az előzővel. A bővítő szó viszont másképp épül fel. Az alsó byte egy 8 bites címkülönbözőséget jelöl, ami előjelhelyesen kerül hozzáadásra. A felső byte-ban arra vonatkozó adatok vannak, hogy melyik indexregiszterről (cím- vagy adatregiszterről) van szó, milyen hosszú az index (szó előjellel vagy kettős-szó), és mi a regiszter száma. Ennek a regiszternek a tartalma (vagy a tartalom egy része) előjelhelyesen kerül hozzáadásra.

Abszolút, rövid:

Ennél a címzési módnál egy olyan bővítő szó van, ami az operandus abszolút, előjelhelyes, 16 bites címét tartalmazza.

Abszolút, hosszú:

Ennél a címzési módnál két olyan bővítő szó van, amelyik az operandus címét tartalmazza. A 32 bites cím magasabb helyiértékű része az első bővítő szóban, az alacsonyabb helyiértékű része a második bővítő szóban van.

Programszámláló (PC) relatív címzése, címkülönbséggel:

Ehhez a címzési módhoz egy bővítő szóra van szükség. Az operandus címe a programszámláló és a bővítő szóban levő, előjelhelyes, 16 bites különbségi érték összegéből adódik.

Programszámláló relatív címzése, címkülönbséggel és indexszel:

Ez a címzési mód megegyezik az előzővel. A bővítő szó viszont másként épül fel. Az alsó byte egy 8 bites címkülönbséget jelöl, ami előjelhelyesen kerül hozzáadásra. A felső byte-ban arra vonatkozó adatok vannak, hogy melyik indexregiszterről (cím- vagy adatregiszterről) van szó, milyen hosszú az index (szó előjellel vagy kettős-szó), és mi a regiszter száma. Ez a címzési mód, és a programszámláló relatív címzésmódjának az egyszerűsége teszi lehetővé olyan programok írását, amelyek bármely címtől kezdődően futtathatók. Pontosan a több munkahelyes rendszereknél van jelentősége annak, hogy szabad rendszerterületeket lehessen használni anélkül, hogy ezeket egymáshoz kellene kapcsolni.

Adatok közvetlen címzése:

Az operandus közvetlenül az utasítást követi. Az operandus hosszától függően egy vagy két bővítő szóra van szükség. Byte-műveletek esetén a bővítő szó alacsonyabb helyiértékű byte-ja kerül felhasználásra. Kettős-szó műveletek esetén a magasabb helyiértékű szó az első bővítő szó, az alacsonyabb helyiértékű szó pedig a második bővítő szó helyén van.

1.5 Az utasításkészlet áttekintése

A 68000-esnek 56 assembler utasítása van. Ez más processzorokhoz képest kevés. A 68000-es azonban a 14 címzésmódjának köszönhetően igen nagy teljesítményű. Ha minden egyes utasításnak külön-külön jelölése lenne, akkor a 68000-esnek több mint 1000 utasítása lenne. A 68000-es assembler utasításkészlete különösképpen a moduláris és a magas szintű nyelveken való programozást segíti elő.

A 68000-es egy „valódi" kétcímű gép. Ez azt jelenti, hogy egy műveletnek mind a célja, mind a forrása a tárolóban van. Így a tárolóban az egyik helyről a másikra való eltolások közvetlenül végrehajthatók. A processzorok legtöbbjében egy tárolóhely tartalmát először a processzor valamelyik regiszterébe kell tölteni, majd ezt követően egy újabb utasítással egy másik tárolóhelyre kell átvinni.

A 68000-esnek egy utasítása egy és öt szó, tehát kettő és tíz byte között lehet. Az utasítás hosszát és jellegét az utasítás első szava határozza meg. A 68000-es utasításai szisztematikusan vannak felépítve.

A 68000-es utasításkészlete a következő csoportokba osztható:

- aritmetikai műveletek (egész számok)
- BCD-utasítások
- logikai utasítások
- eltolási és elforgatási utasítások
- bitmanipulációs utasítások
- adatátviteli utasítások
- programvezérlő utasítások

A 68000-es számos utasítása több különböző típusú adatot tud feldolgozni. Így pl. a MOVE utasítással byte-okat, szavakat és kettős-szókat is tud továbbítani. A különböző címzési módok mellett, amelyek az utasítások révén rendelkezésre állnak, különböző hosszúságú operandusok is kezelhetők.

2. JELEK ÉS BUSZOK A 68000-ESBEN

Ebben a fejezetben azt ismertetjük, hogy hogyan működik a hardver egy 68000-est tartalmazó rendszerben. Ehhez leírjuk a CPU egyes buszműveleteinek időbeli lefolyását. Úgy gondoljuk azonban, hogy az egyes időtartamok egzakt (nanoszekundumokban mérhető) értékei a legtöbb olvasó számára érdektelenek. Ezeknek az időtartamoknak az áttekinthető ábrázolásához oldalakat kitevő táblázatokra lenne szükség, mert a CPU-k gyakorlatilag minden egyes változata egymástól kismértékben eltérő időkkel dolgozik. A CPU-knak egy sereg, különböző maximális ütemosztású vagy üzemi hőmérséklet-tartományú verziója létezik. Azért, hogy e könyv kereteit ne feszítsük szét, itt csak utalunk az egyes gyártó cégek adatlapjaira (alternatív megoldásként amúgy is csak e lapok idemásolása jöhetne szóba). Az egyes műveletek alapvető időbeli lefolyását természetesen ismertetjük, tehát, hogy melyik jelnek melyik jelet kell megelőznie, milyen a kapcsolat a rendszer órajele és az egyes jelek között stb.

Az egyes jelek alapvetően LOW-aktívak vagy HIGH-aktívak. [Ehhez némi magyarázat: egy vezetéken egy jel lehet LOW-szintű – a lassan elterjedő magyar fogalmak szerint alacsony szintű vagy alacsonyra állított – és lehet HIGH-szintű, tehát magas szintű vagy magasra állított. Ezen belül lehet egy alacsony szintű (= alacsonyra állított) jel aktív (tehát valamit befolyásol) és lehet passzív (tehát azt a valamit nem befolyásolja). Ugyanez igaz fordítva is. – Fordító megjegyzése] (LOW = alacsony, HIGH = magas.) Mi a könyvünkben használt rövidítések során az egyes állapotok (LOW vagy HIGH) aktív vagy passzív voltát úgy különböztetjük meg, hogy azt az állapotot jelöljük meg külön, amelyben az alacsony (LOW-állapotú jel) aktív.
Ezt a következőképpen fogjuk ábrázolni:

HIGH-aktív	pl. a szinkron busz engedélyezési jele: E
LOW-aktív	pl. a buszhiba jele: nBERR (BERR = Bus-ERRor)

Tehát egy n betű előreírásával jelöljük, hogy a jel alacsony szintű állapotban aktív.

Egy másik példa a Read/Write (olvasás/írás) vezetéken levő jel állapota. Ha a jel állapota ezen a vezetéken LOW (a német kezdőbetű szerint n), akkor ez írásra (Write) való hozzáférést jelöl (nW). Ha a jel állapota HIGH, akkor ez olvasásra (Read) való hozzáférést jelent. A Read/Write vezetéken levő jel tehát a mi jelölésmódunk szerint így fog festeni: R/nW

2.1 A 68000-es jelei és a csatlakozólábak

A 68000-es egyik nagy előnye, hogy a buszon levő jeleket nem kell multiplex üzemmódban működtetni. Multiplex üzemmód alatt az értendő, hogy a CPU ugyanazon csatlakozóját (pin) különböző egységek használják, és hogy valamilyen módon meg kell tudni különböztetni azt, hogy az adott pillanatban a CPU éppen milyen jelet szolgáltat vagy vár. A multiplexitás nagy előnye, hogy viszonylag kevesebb csatlakozóra van szükség. Másrészt a multiplex üzemmód korlátozza a busz maximális sebességét. Emellett a hibakeresés egy multiplex rendszerben sokkal nehezebb, mint egy olyan rendszerben, amelynél a CPU a jeleket statikusan szolgáltatja. Ugyanis a jelek mérésekor minden egyes időpillanatban tudni kell, hogy melyik is az a jel, amit mérünk. Ezenkívül a multiplex rendszerű CPU-knál többnyire még más, külső hardverre is szükség van, mert a külső egységek és a tárolók ritkán igénylik a jeleket úgy, ahogy azokat a CPU éppen szolgáltatja.

A 68000-es ezért nem egy normál, 40 csatlakozójú tokban van elhelyezve. Ehelyett más, egyenként több mint 60 csatlakozólábú tokformák vannak. Az egyes csatlakozólábak a következők szerint csoportosíthatók:

- feszültségellátás és a rendszer órajele
- címbusz
- adatbusz
- aszinkron buszvezérlés
- szinkron buszvezérlés
- megszakítás vezérlése
- buszvonalátadás vezérlése
- rendszervezérlés és funkciókód-kimenetek.

Feszültségellátás és a rendszer órajele

A 68000-es +5 V feszültséggel működik (természetesen földcsatlakozó is van). Más processzorokhoz hasonlóan egyéb tápfeszültségre nincs szüksége. Az órajelbemenet TTL-kompatíbilis (akárcsak a be- és kimenet). Az órajel frekvenciájának állandónak kell lennie. Mindenekelőtt azonban nem szabad, hogy leálljon, mert a processzornak vannak belső, dinamikus regiszterei, amelyek ellenkező esetben elveszítenék a tartalmukat. A 68000-es egyes változatainál különbözők a minimális és a maximális frekvenciák. Van 4, 6, 8, 10 és 12,5 MHz maximális frekvenciával működő 68000-es.

Címbusz (A1...A23)

A 68000-es címei byte-orientáltak. Ez azt jelenti, hogy a tárban levő mindegyik byte-nak saját címe van. A tárban tehát mindegyik szó két – egy páros és egy páratlan – címen helyezkedik el. A 68000-es adatbusza szó szélességű, egy szót tehát, amely egy alacsony és egy magas helyiértékű byte-ból áll, egyetlen hozzáféréssel el tud érni.
Az „A0" címvezeték viszont kívülről nem létezik (ennek a szerepét veszi át az nUDS és az nLDS). A címbusszal 8 megaszó (tehát 16 megabyte) tárolótér címezhető meg. A megszakítások visszaigazolása során az A1...A3 címvezetékek jelzik, hogy az éppen feldolgozott megszakítás milyen prioritású. Ezen idő alatt a többi címvezeték magas szintű. A 68000-es címbusza három állapotú kimenetekkel rendelkezik.

Adatbusz (D0...D15)

A 68000-es adatbusza mind byte, mind szó szélességű adatot tud fogadni. Egy külső egység ki tud váltani egy nem autovektor megszakítást, és mialatt a CPU nyugtázza a megszakítást, jelzi az adatvezetékek számára a megszakításra vonatkozó vektorszámot. A 68000-es adatvezetékei kétirányú, háromállapotú buszvezetékek.

Aszinkron buszvezérlés

Címstrobe (nAS = Address-Strobe)
Ez a jel mutatja, hogy a címbuszon érvényes adat van.

Olvasó/író jel (R/nW = Read-Write)
Ez a jel azt mutatja, hogy a CPU számára olvasásra vagy írásra való hozzáférés esete áll-e fenn.

Adat-strobe-ok (nUDS, nLDS)
Ezek a jelek az R/nW jellel (olvasó/író jellel) az adatbuszt vezérlik. E jelek használatát egy táblázatba foglaltuk össze. Az alábbi rövidítéseket használtuk:

GDB: érvényes adatbitek

R/nW	nUDS	nLDS	D8...D15	D0...D7
xxx	H	H	nincs érvényes	nincs érvényes
H	L	L	GDB 8...15	GDB 0...7
H	H	L	nincs érvényes	GDB 0...7
H	L	H	GDB 8...15	nincs érvényes
L	L	L	GDB 8...15	GDB 0...7
L	H	L	GDB 0...7*	GDB 0...7
L	L	H	GDB 8...15	GDB 8...15*

*: ez az állapot a jelenlegi kiépítettségi helyzetnek felel meg (vö. a mindenkori érvényes adatlapot).

Adatátvitel nyugtázása (nDTACK = Data-Transfer-ACKnowledge)

Ezen a bemeneten annak a kijelzése történik, hogy az adatátvitel befejeződött. A CPU olvasási hozzáférése esetén az adatok átvitele azt követően kezdődik meg, hogy az nDTACK alacsony szintű. Írási hozzáféréskor a CPU az írandó adatokat addig viszi az adatbuszra, míg az nDTACK alacsony szintű.

Az nDTACK jel (mint valamennyi más vezérlőjel) szinkronizálása belülről történik, hogy a CPU egy aszinkron rendszerben megfelelőképpen tudjon működni. Az órajelvezeték magas szintjénél valamennyi vezérlő és adatvezeték lekérdezésre kerül. Az órajel belső tárolása miatt azonban kisebb időbeli eltolódások adódhatnak a jelek lekérdezése és felismerése között. Ha az nDTACK éppen idejében érkezik, akkor a CPU a maximális sebességgel dolgozik. Egyébként a 68000-es várakozási ciklusokat generál, míg az nDTACK, azaz az adatátvitel nyugtázása meg nem érkezik. Az aszinkron busznak ezt a technikáját a nagy rendszerekben egyszerűbben lehet kezelni, mint egy szinkron busz egyébként szokásos technikáját, amelynél a WAIT-bemeneten keresztül lehet a várakozó ciklusokat kiváltani.

Ha egy nem autovektor megszakítást kell elvégezni, akkor (miután a CPU nyugtázta a megszakítást) ugyancsak az nDTACK jelzi, hogy a megszakításvektor a D0...D7 regiszterben van.

Szinkron buszvezérlés

Engedélyezésjel (E)

Ez a jel az engedélyezés általános jele a szinkron perifériák számára. Ez 6 ütemidőn keresztül mindig alacsony (passzív) és 4 ütemidőn keresztül magas (aktív) állapotú. Mivel ezt a ciklust a hozzáférések nem szinkronizálják, egy szinkron hozzáférésre vonatkozóan fix időtartamot nem lehet megadni.

Az ugyanis, hogy most egy szinkron ciklus indul-e el, attól függ, hogy a külső egységről érkező, erre vonatkozó jel még időben érkezik-e ahhoz az időponthoz képest, amikor a CPU éppen megkezdi egy buszvonal elérését, vagy már túl későn ahhoz, hogy megkapja az engedélyező jelet (E). Ezért a szinkron ciklus alapvetően lassúbb, mint az aszinkron. Arra azonban igen egyszerű módszerek vannak, hogy a szinkron üzemmódú külső egységeket is lehessen az aszinkron buszon keresztül működtetni. Ezeket főként akkor kell alkalmazni, ha sok hozzáféréssel kell számolnunk.

Érvényes perifériacím (nVPA = Valid-Periphery-Address)
Ezen a bemeneten azt jelezzük a CPU-nak, hogy egy szinkron buszciklusnak kell következnie, és az adatátvitelt az engedélyező jellel (E) kell szinkronizálni. Ez a bemenet aktiválódik akkor is, ha egy autovektor megszakítás történt.

Érvényes tárolócím (nVMA = Valid-Memory-Address)
Ezt a kimenetet aktiválja a 68000-es azt követően, hogy egy nVPA-jelet kapott. Ezzel azt jelzi, hogy egy buszvonal szinkron eléréséhez van érvényes cím. Az E-kimenet (engedélyezés) következő aktív fázisában megtörténik a szinkron hozzáférés.

Megszakítás vezérlése

Megszakítási prioritás (nIPL0, nIPL1, nIPL2)
Ez a három bemenet egy megszakítást vált ki, és kódolt formában megadják ennek a prioritását. Ha mindhárom bemenet magas (HIGH), akkor ez azt jelenti, hogy sehonnan sem érkezik megszakítási kérelem. Amikor a CPU egy megszakítást tudomásul vesz, akkor ezt nyugtázza, és vagy egy kizárási vektort (Exception-Vektor) – a nem autovektor megszakítások esetén – vagy az arra vonatkozó közlést várja, hogy egy autovektor megszakítást kell végrehajtania. A további részleteket a megfelelő alfejezetek tartalmazzák.

Buszvonal átadásának vezérlése (Bus Arbitration Control)

Buszvonal kérése (nBR = Bus-Request)
Valamennyi külső egység, amely a buszt maszterként használhatja, egy Wired-Or összeköttetésen keresztül kapcsolódik erre a bemenetre. Ez azt jelenti, hogy valamennyi egy Open-Kollektor (nyitott kollektoros) kimenettel kapcsolódik erre a csatlakozóra. Ehhez – a processzor +5 V-os bemenete után elágazva – egy ellenállás, meghatározott nyugalmi szintről gondoskodik. Ha erre a bemenetre egy alacsony szintű jel kerül, akkor ez azt jelzi a 68000-esnek, hogy a hozzá csatlakoztatott egységek egyike buszmaszterként kívánja elérni.

Buszvonal átadása (nBG = Bus-Grant)

Ez a kimenet valamennyi csatlakoztatott egységnek azt jelzi, hogy a processzor a busz vezérlését az éppen futó ciklus befejezése után átadja.

Buszvonal átadásának nyugtázása
(nBGACK = Bus-Grant-ACKnowledge)

Ez a bemenet azt tudatja a processzorral, hogy valamelyik másik egység átvette a busz vezérlését. A konfliktusok elkerülése céljából ezt a jelet csak akkor szabad kiadni, ha az alábbi négy feltétel teljesült:

1) Az nBG-jelnek (buszvonal átadása) már előzőleg be kellett érkezni.
2) Az nAS-jelnek inaktívnak kell lennie. Ez mutatja azt, hogy a processzor már nem használja a buszt.
3) Az nDTACK-jelnek (adatátvitel-nyugtázás) inaktívnak kell lennie. Tehát sem a tároló, sem a külső egységek nem használják a buszt.
4) Az nBGACK-jelnek inaktívnak kell lennie, a buszt maszterként már egyetlen másik egység sem használhatja.

Rendszervezérlés és a funkciókód-kimenetek

Funkciókód-vezetékek (FC0, FC1, FC2)

A funkciókód-vezetékekkel jelzi a CPU, hogy a buszon a hozzáférés melyik fajtája van. Ezek a vezetékek teszik azt is lehetővé, hogy a tároló meghatározott területeit a felhasználó számára elérhetetlenné tegyük. A hozzáférési módok kódolása a következő:

FC2	FC1	FC0	A ciklus típusa
L	L	L	nincs meghatározva
L	L	H	felhasználói (user) állapot, adatok
L	H	L	felhasználói (user) állapot, program
L	H	H	nincs meghatározva
H	L	L	nincs meghatározva
H	L	H	supervisor-állapot, adatok
H	H	L	supervisor-állapot, program
H	H	H	megszakítás nyugtázása

Buszhiba (nBERR = Bus-ERRor)

Ezen a bemeneten azt lehet jelezni a processzor számára, hogy az éppen futó buszciklus közben nehézségek merültek fel. A lehetséges okok:

- Hiba a megszakítás kezelésében.
- Valamelyik egység nem válaszol. Ezt a hibát egy úgynevezett Watchdog-Timer is kiválthatja. Ha egy bizonyos idő eltelte után a buszciklus nem egy nDTACK jellel fejeződik be, akkor az időzítő a ciklust egy buszhibával megszakítja.

- Nem megengedett tárolótérhez való hozzáférési kísérlet, amit pl. egy MMU (memóriakezelő egység) állapított meg.
- Más egyéb hibaforrás, az alkalmazástól és a kiépítettségi foktól függően.

A buszhibajel és az álljjel egymással funkcionálisan összefüggnek. A kettő együttesen állapítja meg, hogy a hibás buszciklus megismétlődjön-e, vagy hogy egy buszhibakizárás történjen-e. További információk a rendszer vezérléséről szóló alfejezetben találhatók.

nHALT

Ez egy kétirányú vezeték. Ha ez a vezeték aktívvá válik, akkor a processzor az éppen futó ciklus befejezését követően megáll. Ekkor valamennyi vezérlőjel inaktívvá válik, és valamennyi háromállapotú vezeték a nagyohmos állapotba kerül.

Ha a processzor a program feldolgozását – pl. egy kettős buszhiba miatt – megszakítja, akkor ezt a vezetéket aktiválja, és ezen keresztül jelzi valamennyi külső egység számára az állapotát.

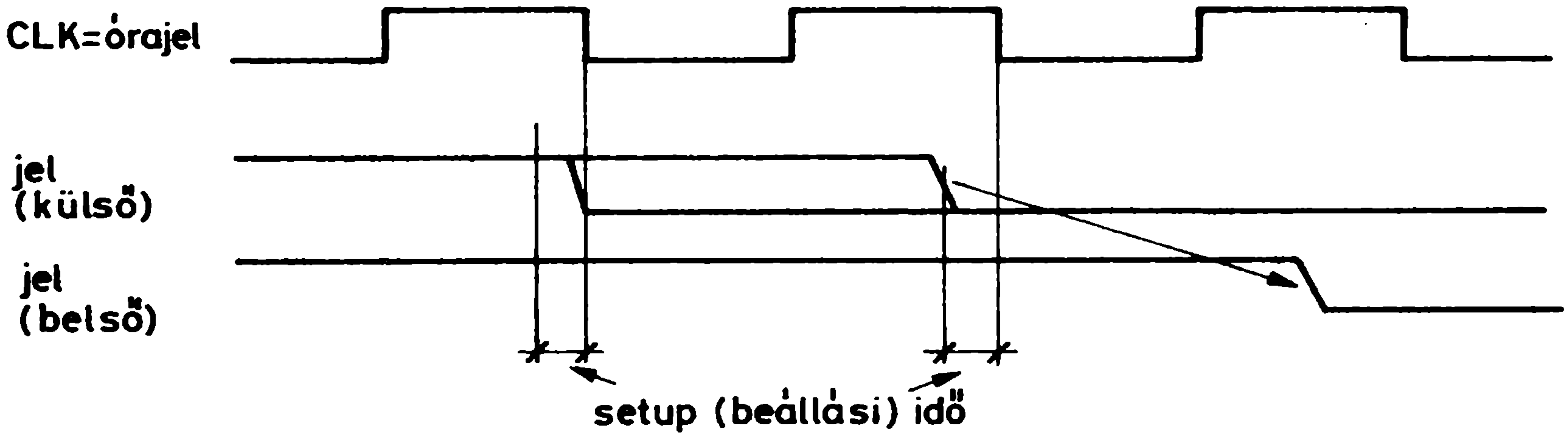
nRESET

A CPU e csatlakozója is kétirányú. Egy külső RESET-jellel a processzor működése visszaállítható, és ezáltal elindítható a rendszer inicializálása. Belülről a RESET-utasítás kiadásával lehet kiváltani. Ekkor ez a csatlakozó kimenetként működik, és 124 ütemidőn át alacsony állapotú. Ezalatt az ezzel a vezetékkel összekapcsolt külső egységek visszaállnak az eredeti állapotukba. Ez a 68000-esre nem vonatkozik, mert ez a feldolgozást változatlan regisztertartalmakkal az éppen következő utasítás végrehajtásával folytatja. Ez a képessége arra használható, hogy egy hiba bekövetkezése után a külső egységeket újra lehessen inicializálni.

Az aszinkron bemeneteket (pl. nVPA, nIPL0,...nIPL2 vagy az nDTACK) a 68000-esnek kell szinkronizálnia. Erre a célra a 68000-es azért ezeket a rendszer órajelének (CLK) lefutó élére minden alkalommal leolvassa. Ahhoz, hogy a jeleket helyesen lehessen felismerni, ezeknek a jeleknek már a leolvasását megelőző bizonyos időben „élniük” kell. Ezek az idők a 68000-es különböző változatainál eltérők.

A 68000-es változatai	Beállási (setup) idő (Aszinkron bemenetek)
4 MHz	min. 30 ns
6 MHz	min. 25 ns
8 MHz	min. 20 ns
10 MHz	min. 20 ns
12,5 MHz	min. 20 ns

A processzor az összes többi bemenet állapotát ugyanígy a rendszer órajelének lefutó élére veszi át. Ezért ezeknek a jeleknek is ebben az időben stabilan élniük kell. Lényegében ezekre a jelekre is a táblázatban megadott értékek vonatkoznak. Az egyes jelek elvben nem szűnhetnek meg addig, amíg a processzor az illető ciklust nem fejezte be.



A külső és a belső jelek közötti kapcsolat

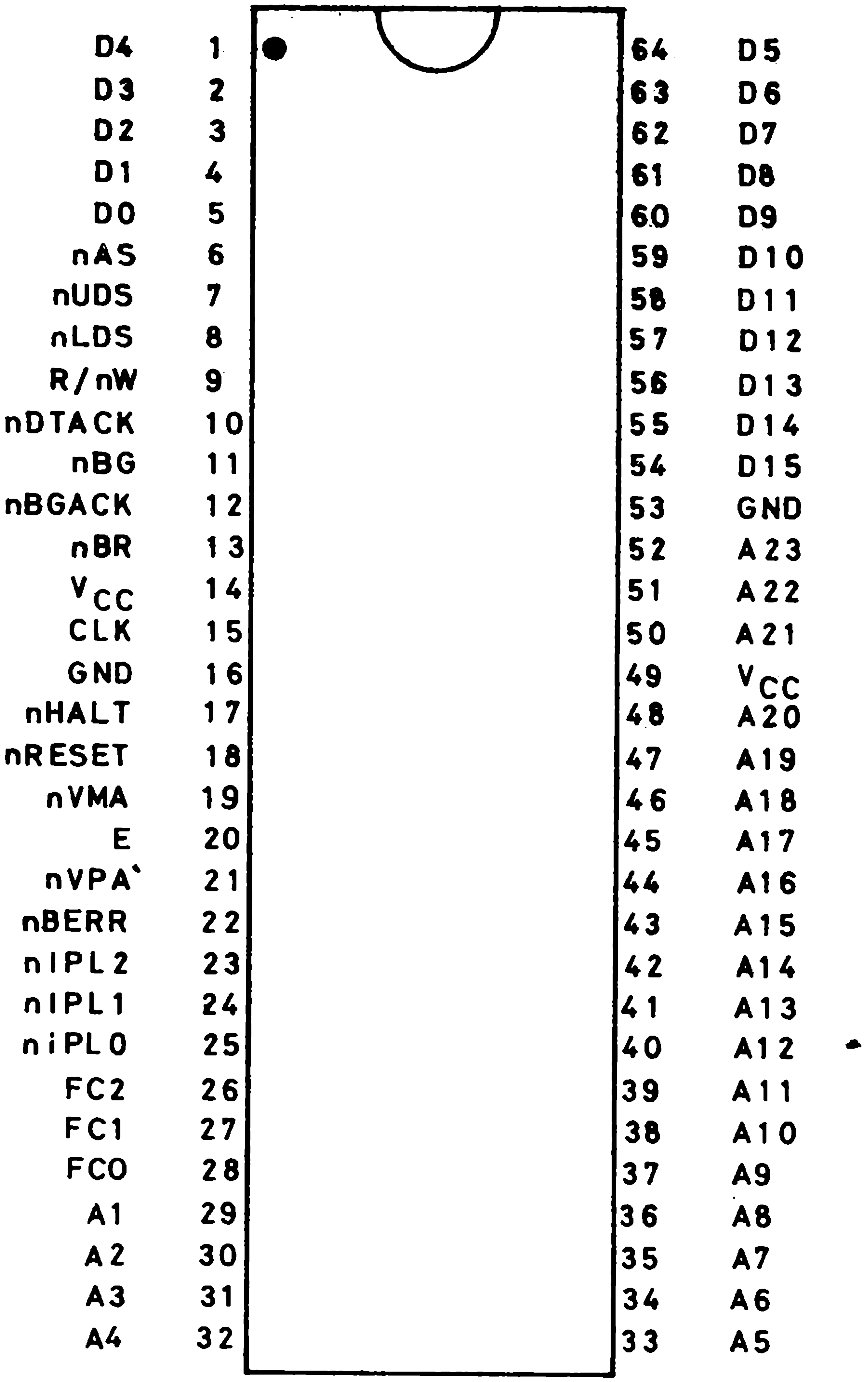
A következő táblázat még egyszer bemutatja a 68000-es jeleit:

Jelölés	Bemenet/ Kimenet	Aktív H/L	Három-állapotú	Funkció
A1...A23	K	H	igen	címvezetékek
D0...D15	B/K	–	igen	adatvezetékek
nAS	K	L	igen	cím-strobe
R/nW	K	H/L	igen	olvasás/írás
nUDS nLDS	K	L	igen	engedélyezés felső alsó
nDTACK	B	L	–	adatátvitel nyugtázása
nBR	B	L	–	buszvonal kérése
nBG	K	L	nem	buszvonal átadása
nBGACK	B	L	–	buszvonal átadásának nyugtázása
nIPL0, nIPL1, nIPL2	B	L	–	megszakítási bemenetek
nBERR	B	L	–	buszhiba
nHALT	B/K	L	*	állj
nRESET	B/K	L	*	RESET
E	K	H	nem	szinkron ciklus engedélyezése
nVPA	B	L	–	érvényes cím/szinkron ciklushoz
nVMA	K	L	igen	érvényes tárcím (szinkron ciklushoz)
FC0,				

FC1, FC2	K	H	igen	funkciókód
CLK	B	H	–	rendszer órajele
Vcc	–	–	–	tápfeszültség (+5 V)
GND	–	–	–	föld (0V)

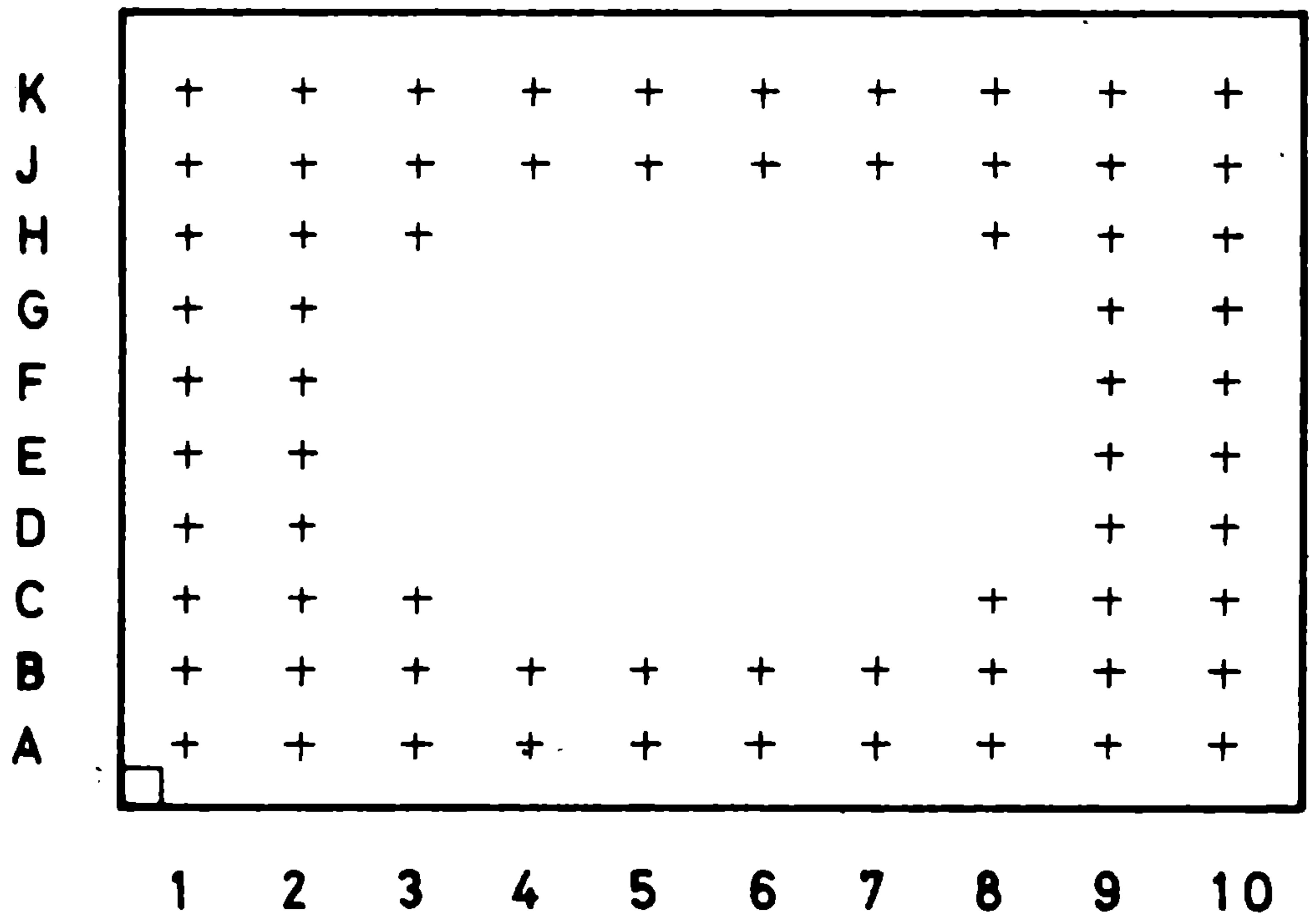
* Open-Kollektor (nyitott kollektoros) kimenet, nem háromállapotú kimenet.
[A Bemenet rövidítése a táblázatban = B, a Kimenet rövidítése a táblázatban = K. – Fordító megjegyzése]

A Dual-In-Line-tokozásban levő 68000-es mikroprocesszor lábkiosztása


D4 1
D3 2
D2 3
D1 4
D0 5
nAS 6
nUDS 7
nLDS 8
R/nW 9
nDTACK 10
nBG 11
nBGACK 12
nBR 13
V_{CC} 14
CLK 15
GND 16
nHALT 17
nRESET 18
nVMA 19
E 20
nVPA* 21
nBERR 22
nIPL2 23
nIPL1 24
niPL0 25
FC2 26
FC1 27
FC0 28
A1 29
A2 30
A3 31
A4 32
64 D5
63 D6
62 D7
61 D8
60 D9
59 D10
58 D11
57 D12
56 D13
55 D14
54 D15
53 GND
52 A23
51 A22
50 A21
49 V_{CC}
48 A20
47 A19
46 A18
45 A17
44 A16
43 A15
42 A14
41 A13
40 A12
39 A11
38 A10
37 A9
36 A8
35 A7
34 A6
33 A5

(felülnézetben ábrázolva)

A PIN–GRID tokozásban levő 68000-es mikroprocesszor lábkiosztása



(alulnézetben ábrázolva)

láb	funkció
A1	nincs bekötve
A2	nAS
A3	D1
A4	D2
A5	D4
A6	D5
A7	D7
A8	D8
A9	D10
A10	D12
B1	nDTACK
B2	nLDS
B3	nUDS
B4	D0
B5	D3
B6	D6
B7	D9
B8	D11
B9	D13
B10	D15
C1	nBGACK
C2	nBG
C3	R/nW
C8	D14
C9	A23
C10	A22

láb	funkció
D1	nBR
D2	V_{cc}
D9	GND
D10	A21
E1	CLK
E2	GND
E9	V_{cc}
E10	A20
F1	nHALT
F2	nRESET
F9	A18
F10	A19
G1	nVMA
G2	nVPA
G9	A15
G10	A17
H1	E
H2	nIPL2
H3	nIPL1
H8	A13
H9	A12
H10	A16
J1	nBERR
J2	nIPL0
J4	nincs bekötve
J5	A2

láb	**funkció**
J6	A5
J7	A8
J8	A10
J9	A11
	A14
K1	nincs bekötve
K2	FC2
K3	FC0
K4	A1
K5	A3
K6	A4
K7	A6
K8	A7
K9	A9
K10	nincs bekötve

2.2 Aszinkron buszhozzáférés

Az aszinkron hozzáférés a processzornak a tárolóhoz vagy a külső egységekhez való normál hozzáférési módja. Az aszinkron azt jelenti, hogy egy buszciklusnak, tehát a tárolóhoz vagy egy külső egységhez való komplett hozzáférésnek, a hossza nem rögzített, valamint nem egy meghatározott időpontban kell indulnia (mint a 6800-as vagy a 6502-es CPU-jánál). A külső egységek (tárolók és perifériák) minden olyan alkalommal leadnak egy jelet a processzornak, amikor a hozzáférést befejezték. Ha egy ciklus során hiba lép fel, akkor ezt is jelzik a 68000-es számára. Ezt a technikát könnyebben kézben lehet tartani, mint más, rugalmas buszhozzáférési időkkel dolgozó módszereket. A 68000-esnél különösen a buszhibák felismerését lehet könnyen megoldani.

Egy buszciklusnak a rendszerórajel (CLK) minimum négy ütemciklusára van szüksége. Ez a négy ütemidő további nyolc félütemre osztható, amelyeket „S0...S7”-tel jelölünk. Egy 8 MHz-es 68000-es rendszernél egy buszciklus időtartama tehát minimum 500 ns. Mindegyik buszciklus (adatátviteli ciklus) tetszőleges időtartamra bővíthető.

Három aszinkron buszciklus között teszünk különbséget.
Ezek:

1) az olvasási ciklus
2) az írási ciklus
3) a Read-Modify-Write (olvasási-módosítási-írási) ciklus.

Az olvasási ciklus

Egy olvasási ciklus során adatokat hozunk be a tárolóból vagy más külső egységből a 68000-esbe. Egy olvasási ciklus alatt a 68000-es vagy egy byte-ot (8 bit), vagy egy szót (16 bit) tud betölteni. A kettős-szó hosszúságú (32 bit) adatokat a 68000-es két, egymást követő ciklusban tölti be. Az adatok mindenkori hosszúságát az utasításban kell megadni.

A 68000-esnek belül van még egy „A0” címe. Egy byte típusú hozzáféréskor az A0 dönti el, hogy a hozzáférés a szó melyik feléhez történjen. Kívülre ezt az nLDS és az nUDS kimenetek jelzik. Ha egy byte hosszúságú hozzáférés egy páratlan címre mutat (A0 = 1), akkor az nLDS lesz alacsony állapotú (LOW), ha a byte hosszúságú hozzáférés páros címre mutat (A0 = 0), akkor az nUDS lesz alacsony.

Az S0...S2 félciklusok alatt a CPU kimenetei kerülnek aktiválásra. Először a funkciókód-vezetékek kerülnek beállításra, azután a címvezetékek mennek át a nagyohmos állapotból az aktív állapotba. Ezt követően az R/nW vezeték, valamint az nLDS és az nUDS kerül beállításra. Ezzel egyidejűleg a CPU az nAS-en keresztül tudatja, hogy a címbuszon érvényes cím van.

Az S3 és S4 félciklusok alatt újabb jelek nem kerülnek a vezetékekre. Most a megszólított egységnek az adatokat a lehető leggyorsabban szolgáltatnia, és ezt a 68000-es számára az nDTACK-kel jeleznie kell.

Az S4 lefutó élétől kezdődően a 68000-es minden egyes órajelciklusban (mindig a CLK lefutó élére) lekérdezi az nDTACK bemenet állapotát. Ha az nDTACK jel felismerése megtörtént, akkor az adatbusz a következő lefutó élre egy belső, közbülső regiszterbe kerül átvitelre, és a ciklus az aktivált kimenetek törlésével befejeződik. Ha ez nem így van, akkor várakozó ciklusok következnek. Mivel az nDTACK lekérdezése az S4-től kezdődik, az adatok átvétele legkorábban az S6 lefutó élére következhet be. Az itt bemutatásra kerülő két idődiagram még egyszer bemutatja a különböző olvasási ciklusokat:

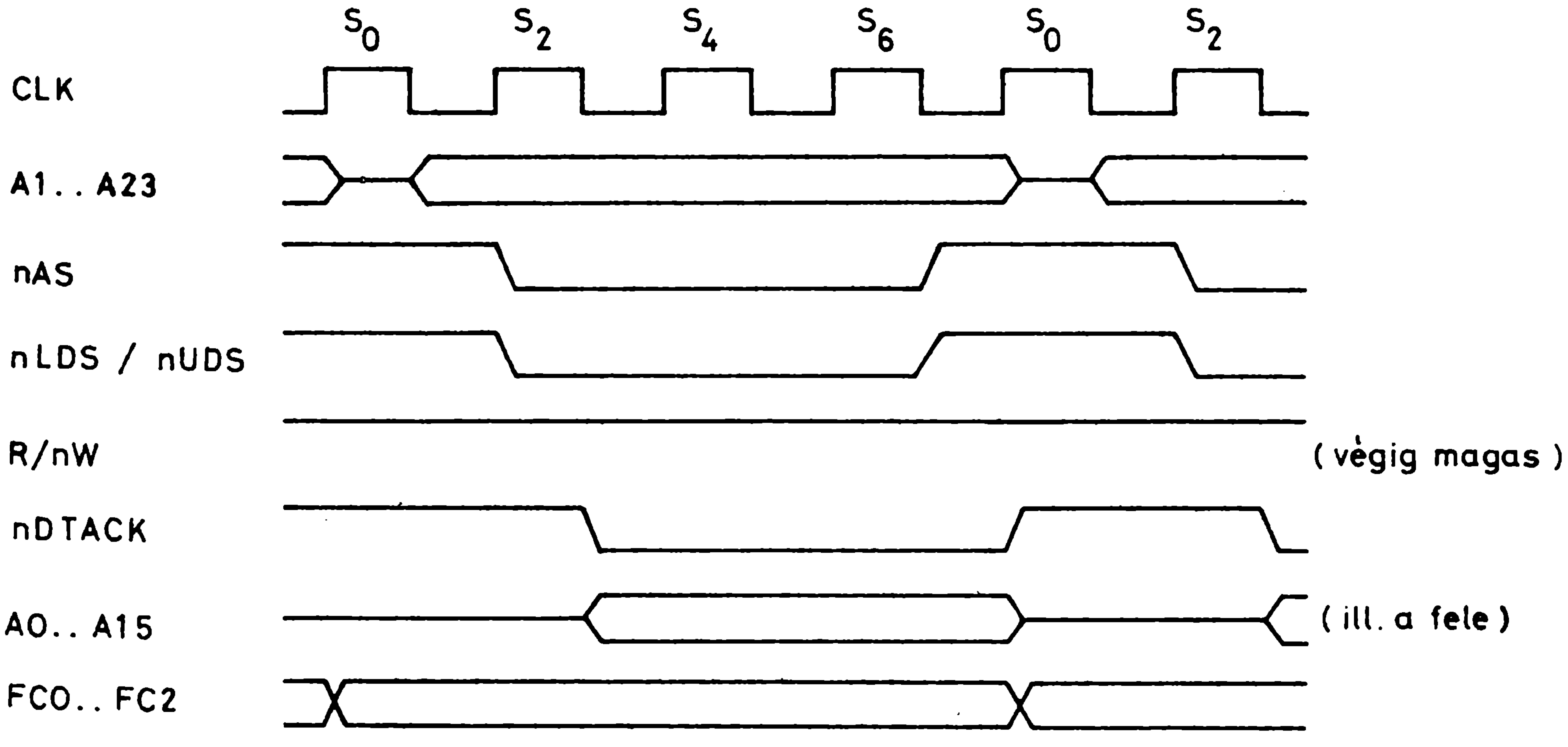
Az írási ciklus

Egy írási ciklus során a 68000-esből adatokat írunk a tárolóba vagy a külső egységekbe. A 68000-es egy ciklus alatt vagy egy byte-ot (8 bit), vagy egy szót (16 bit) tud írni. A kettős-szó hosszúságú adatok írása két, egymást követő ciklusban történik.

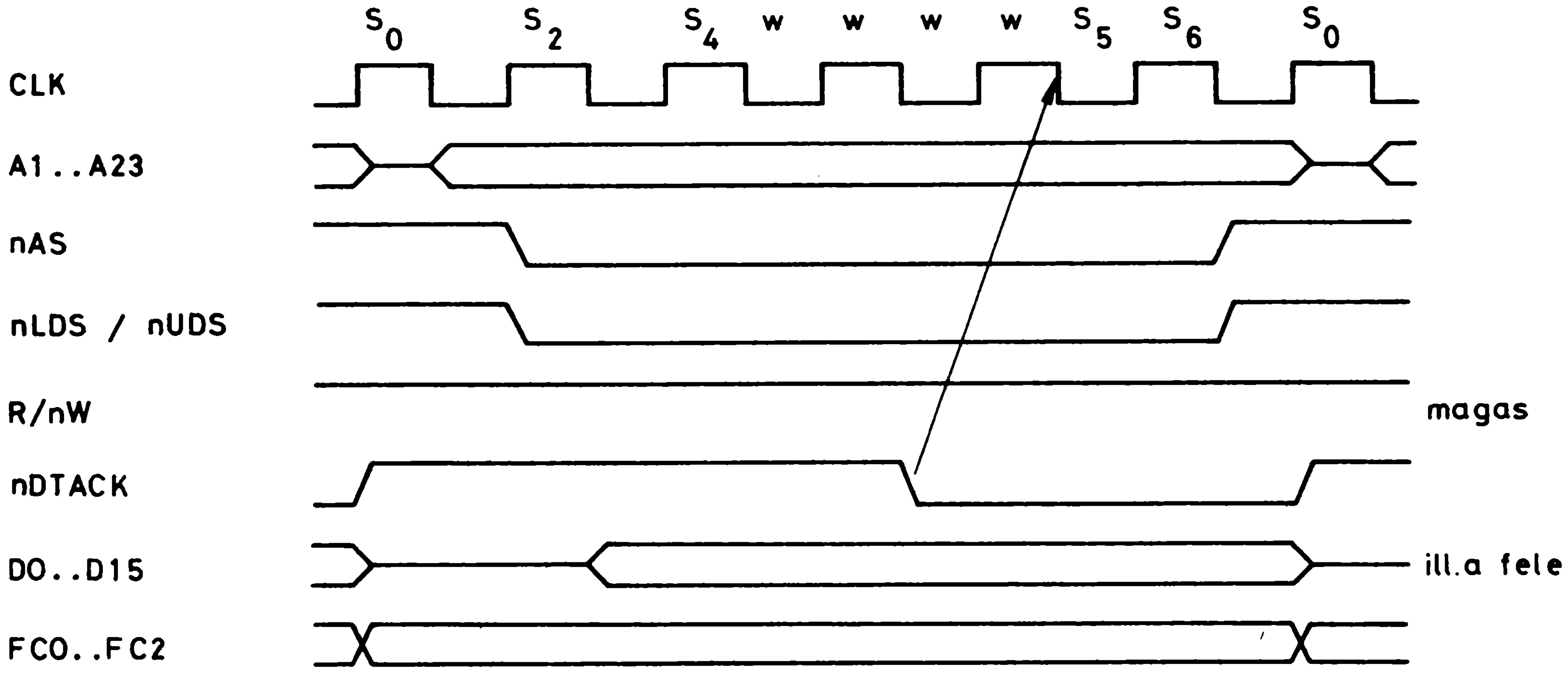
Az írási ciklushoz is a rendszer órajelének minimum négy ütemciklusára van szükség (felosztva az S0...S7 félciklusokra). Egy 8 MHz-es 68000-es rendszernél egy írási ciklus időtartama tehát minimum 500 ns.

Az S0 időpontban (akárcsak az olvasásra való hozzáférésnél) valamennyi vezérlőjel inaktív, a címbusz is nagyohmosra van kapcsolva. Csak a funkciókód-kimenetek állítódnak be a bekövetkezendő ciklustól függően. Az S1 időpontban a címvezetékek a nagyohmos állapotból aktiválásra kerülnek, és így megadják a hozzáférésre vonatkozó címet.

Az S2 időpontban aktiválódik az nAS jel (cím-strobe). Az olvasási ciklussal ellentétben most az R/nW jel is alacsony (LOW) állapotú lesz, és ezzel jelzi az írási ciklust. Azért, hogy az esetlegesen meglevő adatbuszproblémák miatt az átviteli irány átkapcsolásához elegendő idő álljon rendelkezésre, az adatbusz a nagyohmos állapotból csak az S3 időpontban kerül az aktív állapotba. Végül az S4 felfutó élére az nLDS és nUDS jelek a hozzáférés formátumától függően aktiválásra kerülnek.


S0
S2
S4
S6
S0
S2
CLK
A1.. A23
nAS
nLDS / nUDS
R/nW
(végig magas)
nDTACK
AO.. A15
(ill. a fele)
FCO.. FC2

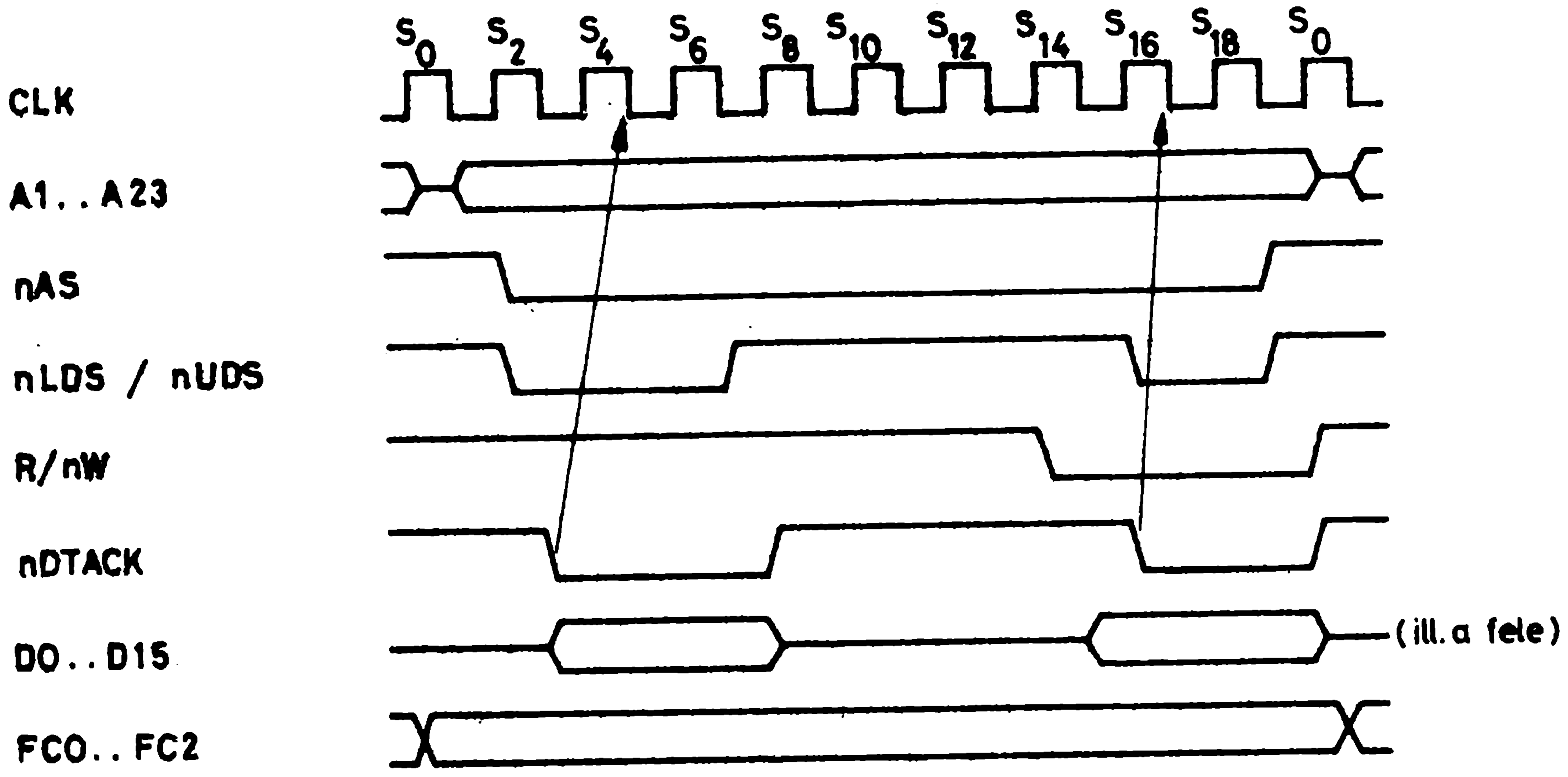
A 68000-es olvasási ciklusa (várakozó ciklusok nélkül)


S0
S2
S4
w
w
w
w
S5
S6
S0
CLK
A1..A23
nAS
nLDS / nUDS
R/nW
magas
nDTACK
DO..D15
ill.a fele
FCO..FC2

A 68000-es olvasási ciklusa (várakozási ciklusokkal)

A megszólított egységnek most az adatokat át kell vennie, és ennek nyugtázásaként az nDTACK vezetéket aktiválnia kell. A nyugtázás átvétele az olvasási ciklushoz hasonlóan történik. A várakozási ciklusok az olvasási ciklushoz hasonlóan az S4 félciklus után kerülnek kiváltásra. A ciklus végén a CPU az adatvezetékeket és a címvezetékeket újra visszakapcsolja a nagyohmos állapotukba.

A következő diagram az írási ciklus menetét ábrázolja:


S0
S2
S4
S6
S8
S10
S12
S14
S16
S18
S0
CLK
A1..A23
nAS
nLDS / nUDS
R/nW
nDTACK
DO..D15
(ill.a fele)
FCO..FC2

A 68000-es írási ciklusa

A Read-Modify-Write (olvasási-módosítási-írási) **ciklus**

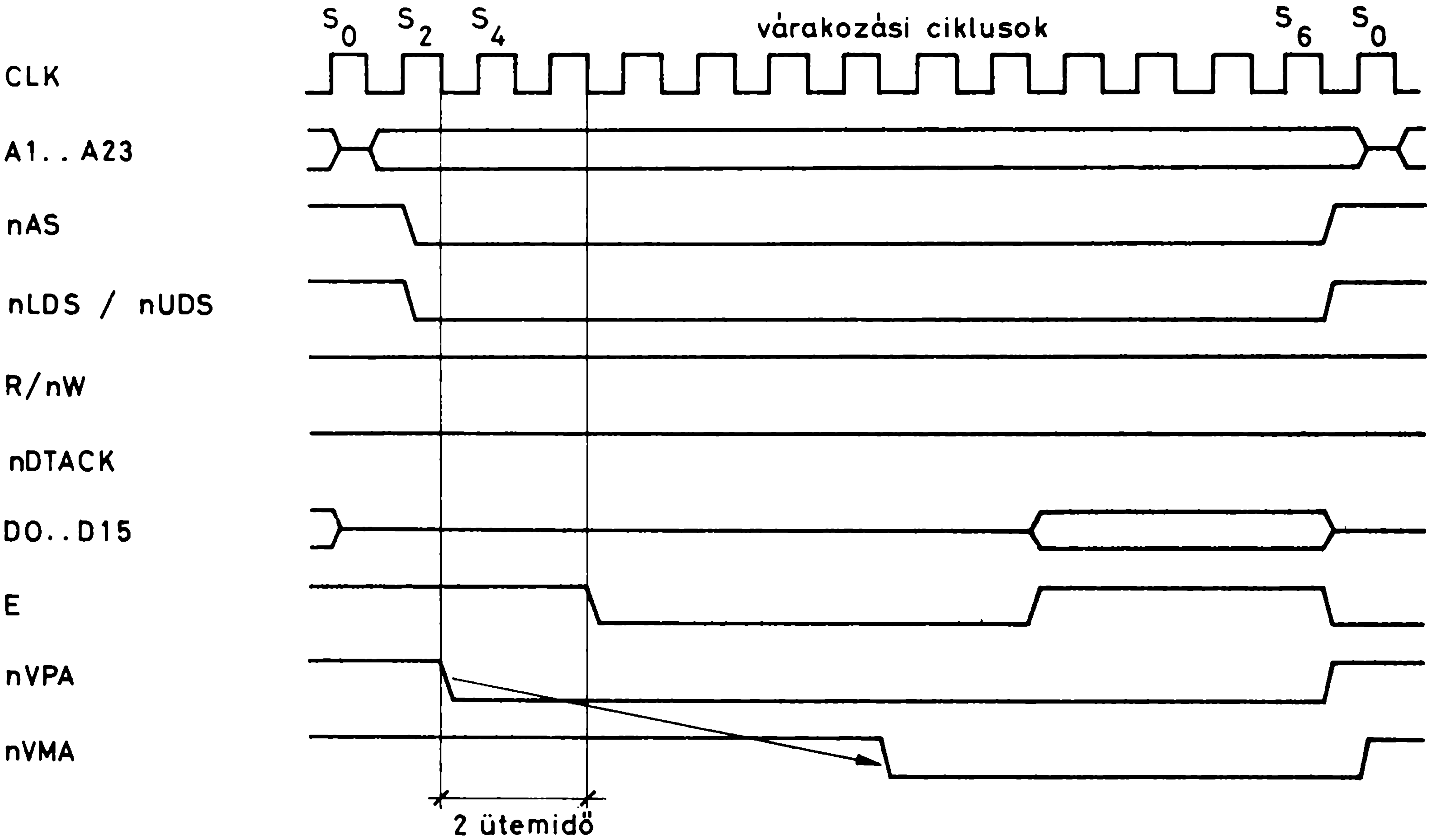
Egy olvasási-módosítási ciklusban először adatokat olvasunk be a CPU-ba, ezeket a CPU feldolgozza, és a módosított adatokat ugyanarra a címre visszaírjuk, ahonnan elolvastuk. A folyamat menetét már maga az „olvasás-módosítás-írás" megnevezés – Read-Modify-Write –leírja. Az olvasási-módosítási-írási ciklus annyiban különbözik a többi ciklustól, hogy az adatokat ugyanarra a címre írjuk vissza, ahonnan azokat kiolvastuk.

A 68000-esnél egy olvasási-módosítási-írási ciklusnak két módozata létezik. Az első módozat az olyan utasításoknál, mint pl. ROL, ROR, LSL és LSR egy tárolóhelyen kerül végrehajtásra. Ebben a módozatban a processzor először beolvassa az operandust, annak tartalmát megváltoztatja, és a megváltoztatott tartalmat ugyanarra a helyre visszaírja, ahonnan azt elővette. Eközben a processzor egyik regiszterének a tartalma sem változik meg. Az olvasási-módosítási-írási ciklus e módozata ugyanúgy megy végbe, mint egy normál olvasási ciklus, amit egy írási ciklus fejez be. Az egyes ciklusok között az nAS rövid időre inaktívvá válik. Így lehetőség van arra, hogy az olvasási-módosítási-írási ciklus e módozatát egy buszvonal iránti kérelem megszakítsa.

Az olvasási-módosítási-írási ciklus másik módozatánál nincs lehetőség a megszakításra, mert az nAS jel végig aktív állapotú. Ezt a módozatot csak TAS (Test-And-Set) utasítás használja. Ez az utasítás a többprocesszoros rendszerekben való kommunikációra használatos. Ehhez csak byte hosszúságú adatok használhatók. Ezért ilyenkor az adatbusznak mindig csak az egyik fele kerül felhasználásra, tehát vagy a D0...D7 rész (amit az nLDS), vagy a D8...D15 rész (amit az nUDS) jelöl.

Az olvasási-módosítási-írási ciklus e második módozatának menetét az áttekinthetőség céljából csak címszavakban ismertettük. Az S7 időpontig az olvasási-módosítási-írási ciklus lefolyása pontosan olyan, mint egy közönséges olvasási ciklusé. Magának az nDTACK-kel való nyugtázásnak a technikája ugyanaz, mint a közönséges ciklusoké. E téma befejezéseként álljon itt egy ciklusdiagram:

Processzor	**Külső egység**
Címzés R/nW vonalat olvasásra állítani funkciókódot betölteni címet behozni cím-strobe-ot aktiválni nUDS-t vagy nLDS-t aktiválni	
	Adatátadás címet dekódolni adatokat a D0...D7-re vagy a D8...D15-re átadni nDTACK-ot aktiválni
Adatátvétel adatokat ideiglenesen tárolni nLDS-t, ill. nUDS-t visszaállítani adatmódosítást megkezdeni	
	Ciklust lezárni adatokat kikapcsolni nDTACK-ot visszaállítani
Írás megkezdése R/nW jelet írásállapotra helyezni adatbuszt aktiválni nLDS-t vagy nUDS-t aktiválni	
	Adatátvitel adatokat tárolni nDTACK-ot aktiválni
Adatátvitelt lezárni nLDS-t, ill. nUDS-t visszaállítani nAS-t visszaállítani adatbuszt kikapcsolni R/nW vonalat olvasásra kapcsolni	
	A ciklus lezárása nDTACK-ot visszaállítani


S0
S2
S4
várakozási ciklusok
S6
S0
CLK
A1..A23
nAS
nLDS / nUDS
R/nW
nDTACK
DO..D15
E
nVPA
nVMA
2 ütemidő

Az olvasási-módosítási-írási (Read-Modify-Write) ciklus

A hozzáférések nem választhatók szét! (az nAS végig aktív)

2.3 Szinkron buszhozzáférés

Amikor egy gyártó egy új mikroprocesszorcsaládot fejleszt ki, akkor ritkán áll annyi fejlesztő a rendelkezésére, hogy magának a mikroprocesszornak a kifejlesztése mellett még a szükséges perifériaegységek kifejlesztésével is tudjon foglalkozni. Ennek az a következménye, hogy az új processzor csak nehezen tud piacot találni, mivel ezzel az új processzorral csak nehezen lehet számítógéprendszereket építeni.
A Motorola a 68000-es számára megtalálta annak a lehetőségét, hogy ehhez a processzorhoz a 6800-as (vagy a 65xx-ös sorozat) perifériaegységeinek teljes választékát alkalmazni lehessen. Ezek a perifériaegységek szinkron buszhozzáféréssel működnek.
A 68000-es képes arra, hogy szinkron buszhozzáférést hajtson végre. Az ehhez szükséges szinkronizálási ütemidőt (amelynek a 68xx családban is „E” a neve) a rendszer órajeléből állítja elő. A 8 bites perifériaegységek maximális frekvenciája 2 MHz. Ezért úgy döntöttek, hogy a 68000-es rendszer órajelét 10-zel osztják. Ez a szinkronütem folyamatosan megy, ami azt jelenti, hogy nem akkor kell aktiválni, amikor a szinkron ciklus felismerése megtörténik. Ezt a követelményt támasztották többek között azon perifériaegységekkel szemben is, amelyek a belső időzítőikhez órajelforrásként ezt az órajelet használják. Az E szinkron ütemidő állapota 6 rendszer-ütemcikluson át mindig alacsony (inaktív), azután 4 rendszer-ütemcikluson át mindig magas (aktív).

Az 68000-es egy buszvonalhoz való hozzáférést, általában mindig egy aszinkron ciklussal kezd meg. Az nVPA bemeneten keresztül lehet jelezni számára, hogy szinkron ciklust kell feldolgoznia. Az nVPA jel kívülről igen könnyen előállítható. Mindössze arra van szükség, hogy a szinkron buszhozzáférésű külső egységek kiválasztására szolgáló címdekódolóból a chipkiválasztó jeleket egy VAGY kapun keresztül az nVPA bemenetre adjuk. Mindannyiszor, ahányszor egy szinkron buszhozzáférésű egység kerül kiválasztásra, így lehet tudatni a CPU-val a szinkron ciklust.

A nVPA-jel felismerése után a CPU várakozó ciklusokat iktat be, hogy az „E” szinkronütemet a buszciklussal szinkronizálja. A várakozó ciklusok közben, körülbelül két rendszerciklusidővel azt megelőzően, hogy az E ismét aktív lenne, a 68000-es aktiválja az nVMA-kimenetet, és így jelzi, hogy az E következő aktív állapota a szinkron hozzáférést jelenti. Az nVMA a perifériaegységek oldaláról közvetlenül a chipkiválasztó bemenetekkel vagy a perifériaegységek címkódolójával kapcsolódik össze.

Mivel az E szinkronütemidő szabadon fut, a hozzáférési idők is különbözők. Ezeken belül van legjobbb és legrosszabb eset:

A legrosszabb eset

az nVPA két ütemciklussal korábban, mint amikor az E magas (HIGH) lesz, aktiválódik [ill. négy ütemciklussal azt követően, hogy az E alacsony (LOW) lesz.]

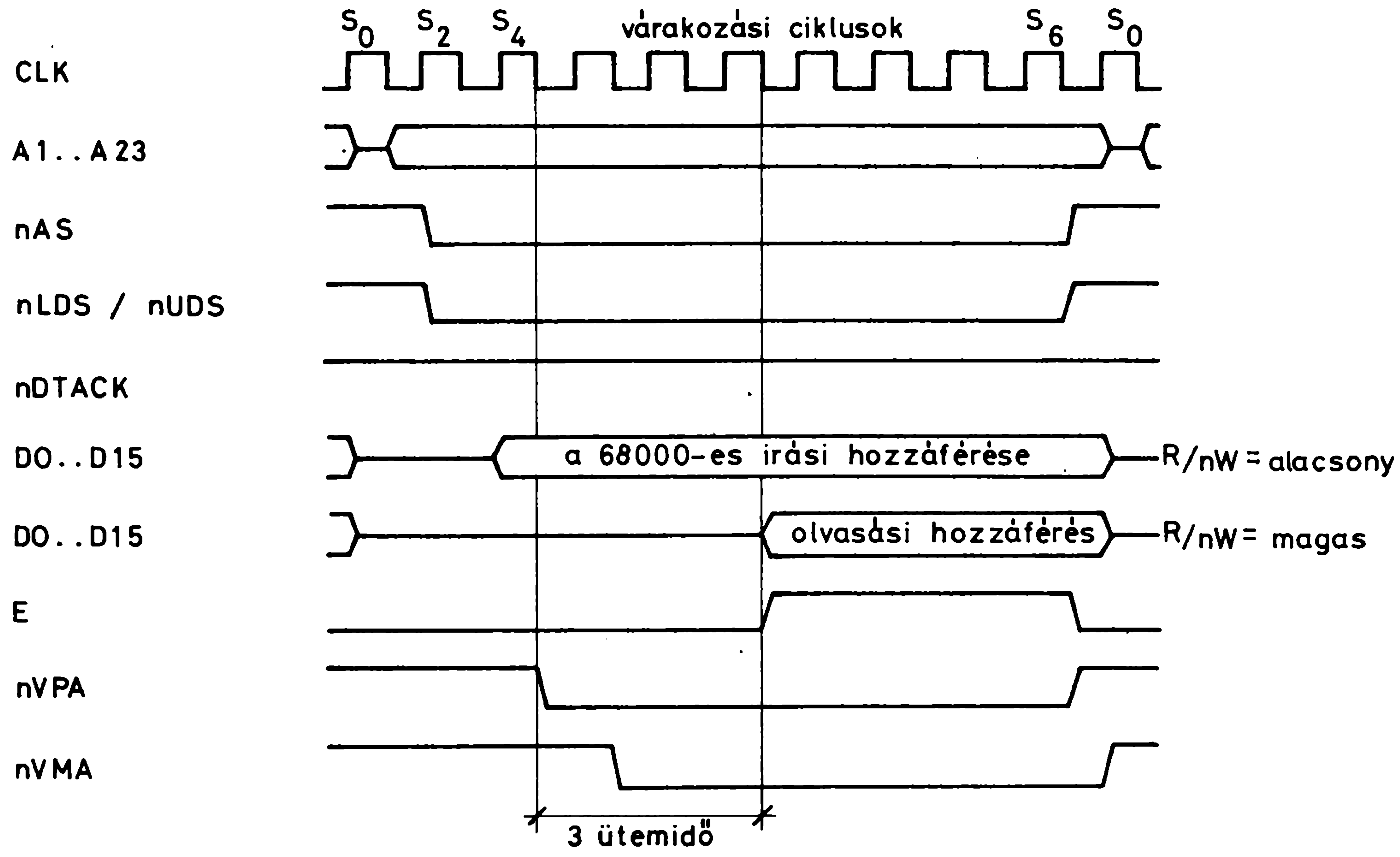
A legjobb eset

az nVPA három ütemciklussal korábban, mint amikor az E magas (HIGH) lesz, aktiválódik [ill. három ütemciklussal azt követően, hogy az E alacsony (LOW) lesz].

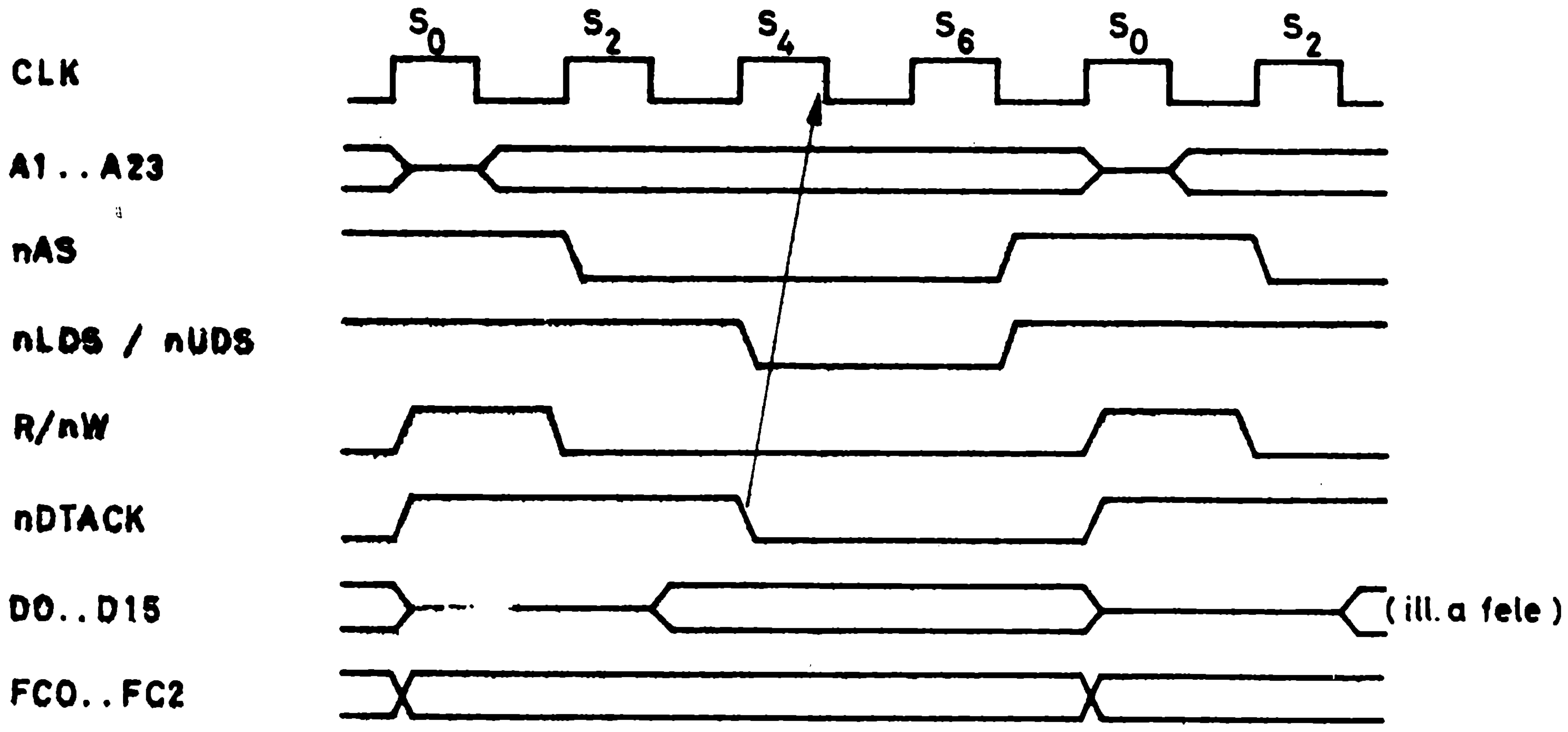
Már ezekből is látszik, hogy nincs arra lehetőség, hogy szinkron buszhozzáférési ciklusokkal a CPU maximális sebességét ki lehessen használni. Ugyanakkor nem szabad elfelejteni, hogy általában csak perifériaegységeket kell szinkron üzemmódban működtetni. A CPU feldolgozási sebessége így összességében alig csökken. Ugyanakkor viszont kis ráfordítással (4 db TTL-szintű IC) arra is van lehetőség, hogy a szinkron üzemmódú egységeket az aszinkron buszon is lehessen működtetni.

A szinkron buszhozzáférés lefolyását három diagramon szemléltetjük. Az első diagram egy tipikus, szinkron olvasási ciklust ábrázol. Az nVPA abban az időpontban kerül aktiválásra, amikor nem valamelyik határesete (legjobb vagy legrosszabb) áll fenn.

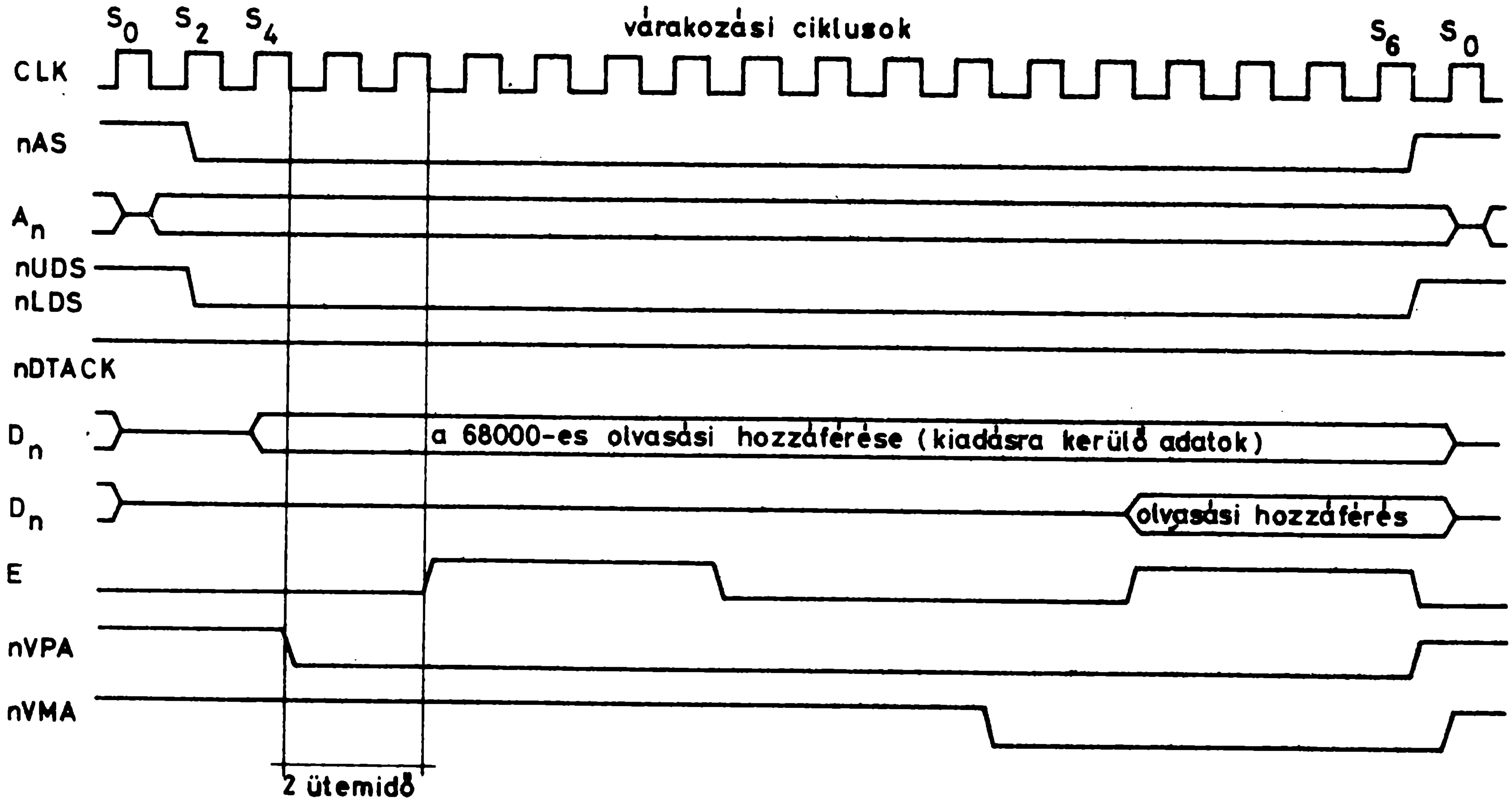
Ezután két diagram következik, amelyek a szinkron buszhozzáférés legjobb és legrosszabb esetére vonatkoznak. Az írási vagy olvasási hozzáférésre ugyanaz a diagram vonatkozik. Az írási vagy olvasási hozzáférés az adatvezetékek időzítésében (timing) különbözik egymástól. Ezért ebbe a két diagramba nem rajzoltuk be a R/nW vezetéket (olvasási-írási vezetékek). Az R/nW lefolyása ugyanolyan, mint az aszinkron buszciklusoknál. Ehelyett az adatvezetékeket kétszer rajzoltuk be, mégpedig egyszer az olvasásra, egyszer pedig az írásra való hozzáférés esetére.


S0
S2
S4
várakozási ciklusok
S6
S0
CLK
A1..A23
nAS
nLDS / nUDS
nDTACK
DO..D15
a 68000-es írási hozzáférése
R/nW = alacsony
DO..D15
olvasási hozzáférés
R/nW = magas
E
nVPA
nVMA
3 ütemidő

A 68000-es tipikus szinkron olvasási hozzáférése


CLK
A1..A23
nAS
nLDS / nUDS
R/nW
nDTACK
D0..D15
FC0..FC2
S_0
S_2
S_4
S_6
S_0
S_2
(ill. a fele)

A legkedvezőbb eset szinkron buszhozzáférés esetén


S0
S2
S4
várakozási ciklusok
S6
S0
CLK
nAS
An
nUDS
nLDS
nDTACK
Dn
a 68000-es olvasási hozzáférése (kiadásra kerülő adatok)
Dn
olvasási hozzáférés
E
nVPA
nVMA
2 ütemidő

A legrosszabb eset szinkron buszhozzáférés esetén

2.4 Megszakítások

Amint erről már írtunk, a 68000-esnél a megszakításokra hét szinten kerülhet sor. A megszakításokat az nIPL0, nIPL1, nIPL2 bemenetek válthatják ki. Egyúttal ezek a bemenetek határozzák meg a prioritások fokozatait is. Az éppen „aktív" prioritási fokozatot az állapotregiszterben az I0, I1 és I2 bitek jelölik. A 1...6 prioritási fokozatú megszakítások maszkolhatók. A megszakításokat a 68000-es tehát csak akkor „akceptálja", ha az állapotregiszerben levő megszakítási maszknak (I1, I2, I3) a prioritási fokozata kisebb. A 7-es fokozatú megszakítás nem maszkolható, tehát mindenkor végrehajtásra kerül (ezért ez a többi megszakítási fokozattal szemben „észreveszi" az órajel lefutó élét). A megszakítások e fokozatát nevezik nem maszkolható megszakításnak (NMI = Non-Maskable-Interrupt). Ezt a megszakítást pl. az válthatja ki, ha kimarad a tápfeszültség, de a regiszterek tartalmát a rendszer összeomlása előtt még a „nem felejtő" félvezető tárakba át kell menteni.

A 68000-es a többi mikroprocesszortól eltérően nem rendelkezik valamennyi megszakítási fokozat részére önálló bemenettel. Ily módon a CPU-n „megspóroltak" néhány csatlakozót. Ebből viszont probléma adódik, mert a három bemenetet gyakorlatilag soha nem lehet pontosan egyidejűleg aktiválni. Fennáll annak a veszélye, hogy emiatt egy „hamis", azaz egy másik prioritási fokozatú megszakítás kerül kiváltásra. Az ilyen jellegű hibák kizárására a megszakítási jel két ütemidőn keresztül el van nyomva, hogy a rendszer egészére vonatkozó zavaró hatásokat messzemenően el lehessen kerülni.

A következő táblázatban a megszakításbemeneteket és az állapotregiszterben levő maszkbiteket állítottuk egymással szembe:

Prioritás	**Az állapotregiszterben levő megszakítási maszk**			**Megszakítási vezetékek**		
	I2	**I1**	**I0**	**nIPL2**	**nIPL1**	**nIPL0**
7	1	1	1	L	L	L
6	1	1	0	L	L	H
5	1	0	1	L	H	L
4	1	0	0	L	H	H
3	0	1	1	H	L	L
2	0	1	0	H	L	H
1	0	0	1	H	H	L
0	0	0	0	H	H	H

Itt a 0-ás prioritás tulajdonképpen nem jelent a szó szoros értelmében megszakítási szintet. Ha mindhárom megszakítási vezeték állapota magas (HIGH), akkor nem kerül sor megszakításra. Ha az állapotregiszterben a megszakítási maszk nulla, akkor valamennyi megszakítás elfogadásra kerül.

A megszakítások felismeréséhez két szabályt kell betartani, hogy a kívánt megszakítást váltsuk ki:

1) A kért megszakítási szintnek annál magasabbnak kell lennie, mint ami az állapotregiszterben maszkolva van (az NMI-t kivéve).

2) A megszakítási jeleknek mindaddig a bemeneteken kell lenniük, amíg ezeket a 68000-es nem nyugtázta.

A 68000-es azzal nyugtázza a megszakításokat, hogy a három funkciókód-kimenetet (FC0...FC2) magasra állítja, és az nAS jelet aktiválja (alacsonyra állítja). Ezen kívül a felismert prioritási fokozat nyugtázásként az A1, A2 és az A3 adatvezetékre kerül. Ezen idő alatt a többi címvezeték állapota magas (HIGH). A 68000-esnél a megszakítások nyugtázását az teszi egyértelművé, hogy valamennyi funkciókód kimenetének az értéke nulla. Ezért ehhez tulajdonképpen a felső címvezetékeket (A4...A23) nem is kell ellenőrizni. A 68010-es és a 68020-as processzoroknál vannak még más üzemmódok is, ezeknél a megszakítás nyugtázása csak akkor történik meg, ha az A4...A23 vezetékek alacsony állapotúak (LOW). Erről többet az erre vonatkozó fejezetben lehet megtudni.

Ha a megszakítás nyugtázása közben a kért megszakítási fokozat megváltozik, tehát változnak a megszakítási bemenetek, akkor a processzor a magasabb fokozatú vektort veszi tudomásul. Ez nagyon jó megoldás, mert így a nagyobb prioritású megszakításokat gyorsabban fel lehet ismerni.

A megszakítások felismerése alapvetően egy utasítási ciklus befejezése után történik. Azoknál az utasításoknál, amelyek egy előzetes adatbehívást (prefetch – MOVEM –) hajtanak végre, az ezt követő utasítás még végrehajtódik. A megszakítás időpontjától függően (a várakozó ciklusok nélkül) a 68000-esnek kb. 45...380 ütemciklusra van szüksége ahhoz, hogy a tulajdonképpeni megszakító program első utasítását feldolgozza.

Egy megszakítás felismerésének a menetét az alábbiak szerint lehet leírni:

1) A megszakítást egy külső egység váltotta-e ki (az IPL0...IPL2 csatlakozókon keresztül).

2) Megvizsgálandó, hogy a prioritás fokozata elegendő-e (tehát, hogy nagyobb-e, mint a maszké), illetve, hogy NMI-(nem maszkolható megszakítás-e). Amennyiben a megszakítás nem megengedett, akkor a CPU a következő utasítást dolgozza fel.

3) Az állapotregiszternek egy belső (kívülről nem hozzáférhető segédregiszterének) tárolása.

4) Supervisor bitet beállítani és a trace-bitet törölni, a megszakítási maszkot aktualizálni (új prioritású fokozat).

5) A programszámláló alacsonyabb helyiértékű szavát a supervisor tárolóban tároljuk.

6) A megszakítás nyugtázása (az FC0...FC2-vel az A1...A3-mal és az nAS jellel), ettől a helytől kezdve a felhasználó a hardveren keresztül három különböző eset között választhat:

   a) az nVPA aktiválódik, ami egy autovektor megszakítást vált ki. Ennek megfelelően betöltődik egy vektor ($19...$1F). Ebben az esetben az nDTACK nem lehet aktív. Mivel a processzor az nVPA aktiválása után az nVMA jelet alacsonyra (LOW) állítja, ügyelni kell arra, hogy nemkívánatos buszhiba ne kerüljön kiváltásra.
   b) Az nDTACK aktiválódik, ami egy nem autovektor megszakítást vált ki. Ezt megelőzően azonban a vektor számát a D0...D7 adatvezetékre kell adni. Ezt az eljárást a 68000-es perifériaegységei használják.
   c) nBERR jel érkezik, ami a processzor számára azt jelzi, hogy nem kell megszakítást kiváltani. Éppen a gépi vezérléseknél fordul elő ismételten, hogy hibás impulzusok megszakításokat váltanak ki. Mivel a perifériaegységek egyike sem váltotta ki ezt a megszakítást, ezt sem az nVPA, sem az nDTACK jel nem nyugtázza. A rendszerbe beépíthető egy ütemidőadó, ami meghatározott idő után egy nBERR jelet szolgáltat. A processzor ezután a $18 számú kizárást (hamis megszakítás) dolgozza fel. Ennek során felülvizsgálható a rendszer, vagy a megszakítás is megismételhető.

7) A vektor címének előállítása a vektor számából (úgy, hogy a vektorszám bitjeit kétszer eltoljuk).

8) A tárolt állapotregiszter tartalmát a supervisor-verembe visszük (ilyenkor egy szó hosszúságú helyet a veremben a programszámláló magasabb helyiértékű szava számára szabadon hagyunk).

9) A programszámláló magasabb helyiértékű szavát a supervisor-verembe töltjük (az alacsonyabb helyiértékű szó és az állapotregiszter közé).

10) Behozzuk a megszakítási program első utasítását, ezután a busz aktiválása nélküli két ütemciklus következik.

11) Behozzuk a megszakítási program második szavát. Erre az időpontra azt is lekérdezzük, hogy van-e újabb megszakítási kérelem. Ha valóban van egy nagyobb prioritású, akkor ehhez először kiváltunk egy kizárást, és ezt követően dolgozzuk fel a korábban bekövetkezett megszakítást. A szokásos módon valamennyi kizárási rutint (tehát a megszakításokat is) az RTE (ReTurn from Exception) utasítás fejez be.

Az autovektor megszakításokat többnyire a 6800-as perifériaegységek használják. A legtöbb 68000-es perifériaegység viszont nem autovektor megszakítási jelet szolgáltat.

A 68000-es perifériaegységeknek van egy megszakítási kimenetük (nIRQ = Interrupt ReQuest) és egy megszakítást nyugtázó bemenetük (nIACK = Interrupt ACKnowledge). Az egységek egyes megszakítási kimenetei egy multiplexeren keresztül (8 a 3-ban, pl. SN 74LS148) a 68000-es megszakítási bemeneteire kapcsolhatók. A processzor megszakítási nyugtázásra is egyszerű módon rákapcsolható a perifériaegységre. Ehhez csak az A1...A3 címvezetékeken levő nyugtázást kell egy demultiplexerrel (3 a 8-ban, pl. SN 74LS138) dekódolni. Ebben az időpontban az nAS-nek alacsonynak, az FC0...FC2-nek és az A4...A23-nak magasnak kell lennie. A demultiplexer kimeneteit a periféria egység mindenkori hozzátartozó nIRQ kimenetével együtt egy VAGY-kapura (két bemenetű, pl. SN 74LS32) kapcsoljuk. A VAGY-kapu kimenetét a perifériaegység mindenkori megfelelő nIACK bemenetével kötjük össze.

Ha egy rendszerben hétnél több megszakítási forrás van, tehát több mint hét prioritási fokozatra van szükség, akkor más módszereket kell alkalmazni. Lehet pl. azt, hogy egy megszakításkor a szoftver annak a forrását ciklikusan lekérdezi (polling), vagy egy megfelelő hardver segítségével, amelyik több prioritási fokozatot ismer (Daisy Chain). A pollingnak az az előnye, hogy hardver-kiegészítésre nincs szükség, viszont lényegesen lassúbb, mint a daisy chain.

A teljesség kedvéért meg kell említeni, hogy vannak olyan 68000-esek, amelyeknél a megszakítások felismerése nem működik helyesen. Ennek tárgyalására azonban nem érdemes kitérni, mert ezeket a Motorola csak 1980 áprilisáig szállította, és csak mintaként, kis darabszámban. Így tehát kizártnak tekinthető, hogy ilyen jellegű, hibás processzorhoz jussunk.

2.5 Buszvonal kérése

Azért, hogy egy mikroszámítógépes rendszerben az adatokat gyorsabban lehessen továbbítani, mint amit a CPU átviteli utasításai lehetővé tesznek, az úgynevezett közvetlen tárhozzáférést (DMA, Direct Memory Access) alkalmazzák.

Az ilyen jellegű átvitelek megvalósításához speciális DMA vezérlőegységek állnak rendelkezésre. Ahhoz, hogy ezeknek az egységeknek valamelyike teljesíteni tudja a feladatát, természetesen át kell vennie az adat-, cím- és a vezérlőbusz feletti ellenőrzését. Ilyenkor buszátvételről (Bus-Arbitration) is beszélünk.

A 68000-esnél a buszátadáshoz három vezeték áll rendelkezésre. Ezek alapvető funkcióját a CPU csatlakozóinak leírásával foglalkozó fejezet elején már ismertettük. Ezen a három vezetéken keresztül a 68000-es handshake kapcsolatot tart a DMA vezérlővel.

Külső hardverrel arra is van lehetőség, hogy a rendszerben több DMA vezérlőt működtessünk. Ugyanerre a célra létezik egy speciális egység, a 68000-es család úgynevezett Bus-Arbitration-Modulja (MC 68452).

A buszátadás azonban nem csak a DMA vezérlő céljait szolgálja. Ez arra is lehetőséget ad, hogy többprocesszoros rendszereket építhessünk. Ilyenkor pl. az egyik processzor hozzá tud férni a másik processzor tárolójához.

A külső vezérlők a buszátadás során előnyben részesülnek, ami azt jelenti, hogy előbb a külső vezérlők valamennyi kérését kell kielégíteni, és csak ezután használhatja a 68000-es a saját buszvonalát.

A 68000-es a buszvonal iránti kérelmet, ami az nBR-en (Bus Request) jön, az nBG-n (Bus-Grant) keresztül a lehető leggyorsabban visszaigazolja. Azt azonban minden esetben kivárja, míg annak az utolsó buszciklusnak, amit maga a 68000-es hajt végre, az nAS jele aktív. Ily módon kerülhetők el a buszkonfliktusok.

Addig amíg a külső egység a buszmaszter, a 68000-es kimenetei nagyohmosak. Ezek alól kivételek a rendszer vezérlőkimenetei, az E szinkronütem és természetesen az nBG kimenet. Az utolsó ciklus után, amit a CPU buszmaszterként végrehajt, a vezetékek a nagyohmos állapotba kerülnek. Ezt csak két ütemciklussal azt követően, hogy az nBGACK (Bus-Grant-ACKnowledge) bemenet újra inaktív (magas, HIGH) lesz, hagyják el, hogy elindulhasson a CPU következő buszciklusa.

A „felkérő" egység addig „használhatja" a buszt, míg azt az nBGACK ismét vissza nem kapcsolja az inaktív állapotába. Ez csak akkor történhet meg, ha az egység a buszciklusait befejezte, és a vezetékek ismét nagyohmosak.

Ha az nBR jel azt megelőzően kapcsolódna ki, hogy egy egység kiadta volna az nBGACK jelet, akkor a 68000-es a normál feldolgozást folytatja. Ez azért van így, hogy az nBR vezetéken levő zavaró impulzusokat figyelmen kívül lehessen hagyni.

Ha az nBR jel aktív marad azt követően, hogy az nBGACK kikapcsolódott, akkor a CPU a buszt azonnal átadja a következő külső egységnek, ő maga nem hajt végre buszhozzáférést. Lehetőség van tehát arra, hogy a különböző egységek sorban várjanak a buszra, és a hozzáférések egymás után történjenek.

Processzor	A „felkérő” egység
	A buszvonal iránti kérelem nBR-t aktiválni
Döntés a buszátadásra vonatkozóan nBG-t aktiválni	
	Visszaigazolás külső hardver határozza meg a buszmasztert a következő buszmaszter vár az éppen futó ciklus végéig nBGACK-t aktiválni nBR-t kikapcsolni
Döntést befejezni nBG-t kikapcsolni és várni az nBGACK kikapcsolására	
	Buszmaszterként dolgozni adatátvitelt lebonyolítani (ugyanazok a szabályok, mint a processzornál) Buszmaszter-tevékenységet befejezni nBGACK-t kikapcsolni
Buszvonalat visszavenni és a következő ciklust megkezdeni (vagy a buszvonalat rögtön a következő maszternek továbbadni)	

A buszvonal átadásának diagramja

2.6 Rendszervezérlés

A rendszervezérlésre a 68000-esnél három bemenet alkalmas. Ennek a három csatlakozónak mindenkor többféle funkciója van.

Az nRESET csatlakozó lehet egyaránt kimenet és bemenet is. Kimenetként a RESET utasítással aktiválható.

Az nHALT csatlakozó is kétirányú. Ha egy kettős busz- vagy címhiba lépett fel, akkor a 68000-es az állj állapotba kerül, mert azt tételezi fel, hogy egy katasztrofális hiba történt. Ebből az állapotából csak egy külső RESET impulzussal lehet kimozdítani. Azon idő alatt, amíg a 68000-es az állj állapotban van, a buszvezetékek nagyohmosak. Az nHALT csatlakozó ilyenkor kimenetként működik, és a CPU számára jelzi az állapotát.

Az nBERR csatlakozó csak bemenet.

A három csatlakozó bemenetként egymással kapcsolatban van:

nRESET	nHALT	nBERR	Hatása
L	L	H	tápfeszültség bekapcsolás, ill. RESET
L	H	H	RESET üzem közben
H	L	H	a 68LLL megállítása
H	L	L	újraindítási (Re-Run) ciklus kiváltása
H	H	L	buszhiba (kizárás)
H	H	H	normál feldolgozás

Amint a táblázatból látszik, a tápfeszültség bekapcsolásakor az nRESET bemenet mellett még az nHALT bemenetet is aktiválni kell. Az nHALT bemenetnek minimum 100 ms időn keresztül alacsonyan kell lennie, hogy egy tiszta RESET jöjjön létre.

Azáltal, hogy a 68000-est az nHALT bemeneten keresztül meg lehet állítani, egy egyszerű hardverrel lehetőség van arra, hogy a 68000-essel mindig csak egy-egy buszciklust hajtassunk végre. Csak arról kell gondoskodni, hogy abban a pillanatban, amikor a processzor az nAS-t aktiválja, az nHALT is aktiválva legyen. Ha a következő buszciklust kell feldolgozni, akkor az nHALT jelet ismét ki kell kapcsolni. A megállási állapotban a vezetékek nagyohmosak (adat- és címbusz) vagy inaktívak (vezérlő vezetékek). Ezt a megállási állapotot nem szabad összetéveszteni egy kettős busz- vagy címhibát követő állj állapottal.

Arra, hogy a processzort egy buszcikluson belül megállítsuk, a perifériaegység nDTACK jele is felhasználható. Ez abban a pillanatban következhet be, amikor a processzornak tovább kell dolgoznia. Egy esetlegesen meglevő Watchdog-timer-ről azonban nem szabad megfeledkezni. Az utasítások egyenkénti végrehajtása a legjobban a 68000-es TRACE-funkciójával valósítható meg.

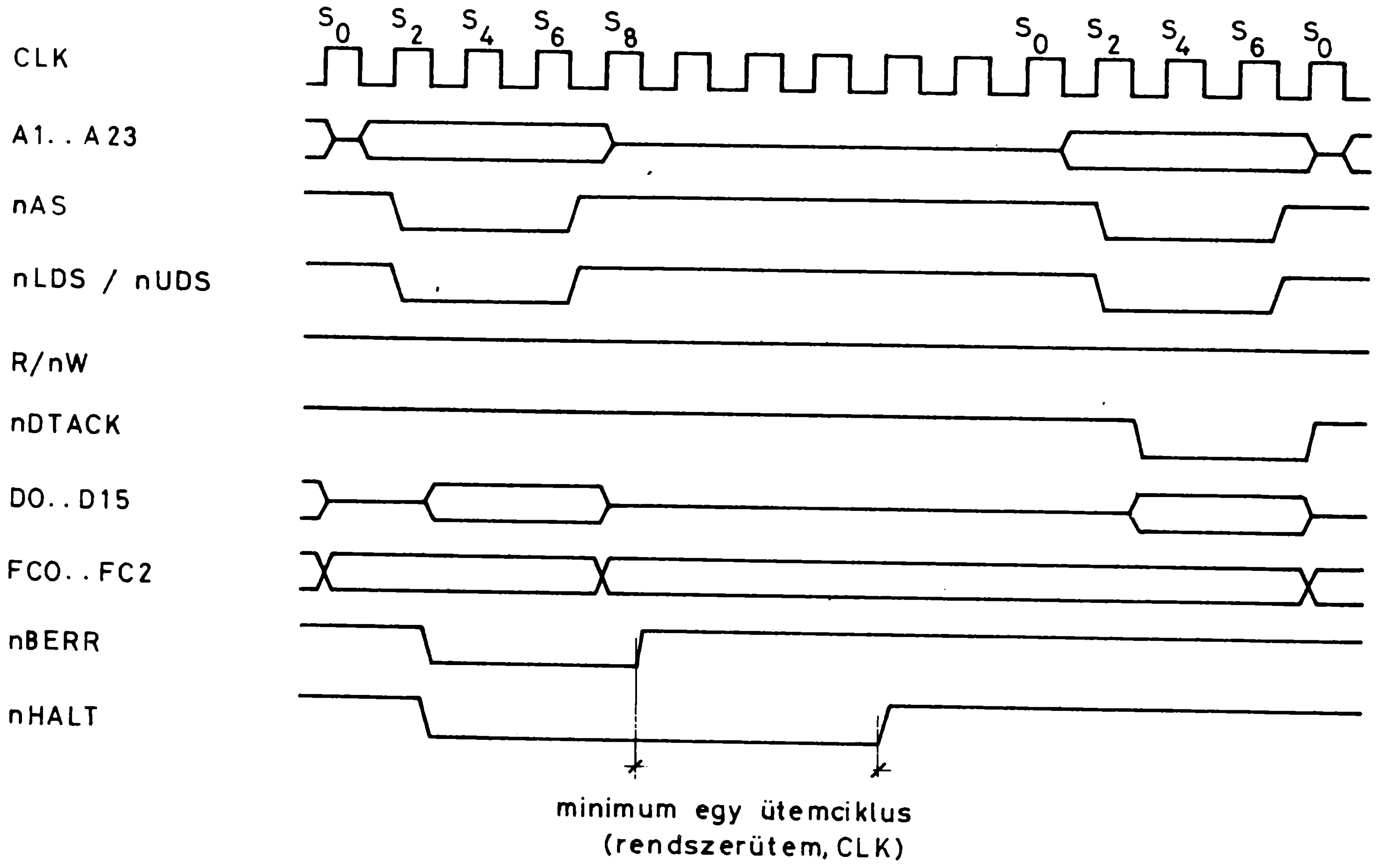
Ha csak az nBERR jel van aktiválva, akkor a 68000-es egy buszhibakizárást indít el. Amikor felismeri az nBERR jelet, befejezi az éppen futó buszciklust, és a buszvonalát a nagyohmos állapotba kapcsolja. Ezt az állapotot csak akkor hagyja el, amikor az nBERR vezeték már nem aktív. A buszhibakizárás feldolgozásának a menetét a 68000-es felépítéséről szóló fejezetben ismertettük.

Ha egy kettős buszhiba vagy akár egy kettős címhiba lép fel, akkor a 68000-es abból indul ki, hogy a normál feldolgozás folytatása már nem lehetséges, ezért megállítja a feldolgozást, és állj állapotba megy. Ebből az állapotból csak egy külső nRESET jel hatására mozdul ki. Az állj állapotot az nHALT csatlakozó jelzi a CPU számára. Lehetőség van pl. olyan hardver összeállítására, ami az állj állapotot jelzi. Ily módon hívható pl. egy kezelő személy, aki a rendszert ismét elindítja, vagy az adatokat egy esetleges javítás előtt kimeneti (pl. segédrendszerekkel). Ismételt buszhiba alapvetően csak kizárási alprogramban léphet fel, hogy ne váltson ki kettős buszhibát.

Ha az nBERR és az nHALT egyidejűleg aktív, akkor a 68000-es az éppen futó ciklust megszakítja abból a célból, hogy megismételje. A két buszciklus között a 68000-es kimenetein aktívak, de csak az adat- és a címbusz van nagyohmosra kapcsolva. A processzor a ciklus megismétlését csak akkor kezdi meg, miután az nBERR és az nHALT ismét inaktív lesz. Az nHALT jelnek az nBERR jel kikapcsolását követően minimum egy ütemidőn belül ki kell kapcsolnia, hogy kifogástalan újrafutási (Re-Run) ciklust lehessen indítani.

Normál esetben az újrafutási ciklus során pontosan ugyanaz a tárhozzáférési művelet kerül elvégzésre. Ez azt jelenti, hogy ugyanahhoz a címhez nyúlunk, hogy ugyanaz a funkciókód, hogy egy írási műveletnél ugyanazokat az adatokat helyezzük el, hogy a vezérlő vezetékeken ugyanazok a jelek vannak. E szabály alól egyetlen kivételt csak a TAS (Test-And-Set) utasítás jelent. Ez egy olvasás-módosítás-írás ciklussal a tárhoz fordul, azonban a két részciklus közben nem szakítható meg (a TAS utasításra többprocesszoros rendszerekben bizonyos rendszerműveletekhez van szükség, hogy az úgynevezett „szemaforok”-at kezelni lehessen). Ha TAS utasítás végrehajtásakor az nBERR és az nHALT egyidejűleg aktív, akkor ennek ellenére csak egy normál buszhibakizárás váltódik ki.

Egy újrafutási ciklus idődiagramja következik, várakozási ciklusok nélkül:


CLK
A1..A23
nAS
nLDS / nUDS
R/nW
nDTACK
DO..D15
FCO..FC2
nBERR
nHALT
S_0
S_2
S_4
S_6
S_8
S_0
S_2
S_4
S_6
S_0
minimum egy ütemciklus
(rendszerütem, CLK)

A 68000-es tipikus újrafutási ciklusa

Alapvetően négy lehetőség van a 68000-es egy buszciklusának lezárására:

Normál lezárás:
az nDTACK a többi vezérlő vezeték előtt minimum egy ütemidővel aktiválódik.

Lezárás álljjal:
az nHALT az nDTACK-kal egy időben, vagy az nDTACK előtt aktiválódik.

Egy buszhibakizárás kiváltása:
az nBERR vagy az nDTACK előtt, vagy ezzel egyidejűleg aktiválódik, és az nDTACK-kel egyidejűleg, vagy ezt követően ismét kikapcsolódik.

Egy újraindítási (Re-Run) ciklus indítása:
az nHALT és az nBERR az nDTACK-kel egyidejűleg, vagy azt megelőzően aktiválódik. Az nHALT jelet az nBERR-t követő minimum egy ütemidőn belül ki kell kapcsolni.

Mivel az szinte lehetetlen, hogy egy buszciklus korrekt lefolyásának menetét a jelek pontosan egyidejűleg történő beállításával minden esetben biztosíthassuk, ezért kell, hogy az nBERR, nHALT és nRESET jeleket ne pontosan a rendszerütem lefutó élében tudjuk változtatni. A 68000-es a bemeneteket a rendszerütemnek alapvetően ebben az időpillanatában kérdezi le. Ha tehát a jeleket egyértelműen ezen élek között aktiváljuk, akkor ezeket a CPU egyidejű jelekként ismeri fel. A jelek azonban abban az időpillanatban is megjelenhetnek, amikor a rendszer ütemjele felfutó fázisban van. A pontos időbeli összefüggésekről a 68000-es mindenkori egyes gyártóinak az adatlapja tud felvilágosítást adni.

A busz vezérlőjeleit azt megelőzően nem szabad kikapcsolni, mielőtt a busz az illető ciklust nem fejezte be.

Az nDTACK, az nBERR és az nHALT jelek aktiválásának a következmnye:

Vezérlőjel	**Az ütem felfutó élére aktiválva**		**Következmény**
	N	**N+2**	
nDTACK	A	S	normál ciklus befejezése
nBERR	nA	x	
nHALT	nA	x	
nDTACK	A	S	normál ciklus befejezése, tartása és folytatása, ha az nHALT kikapcsol
nBERR	nA	x	
nHALT	A	S	
nDTACK	nA	A	normál ciklus befejezése, tartása és folytatása, ha az nHALT kikapcsol
nBERR	nA	nA	
nHALT	A	S	
nDTACK	x	x	ciklus befejezése, és a buszhibakizárás kiváltása
nBERR	A	S	
nHALT	nA	nA	
nDTACK	nA	x	újraindítási ciklus kiváltása
nBERR	A	S	
nHALT	nA	A	
nDATCK	x	x	újraindítási ciklus kiváltása
nBERR	A	S	
nHALT	A	S	
nDTACK	nA	x	újraindítási ciklus kiváltása
nBERR	nA	A	
nHALT	A	S	

A következő rövidítéseket alkalmaztuk:
N : egy párosszámú ütemél száma
A : a jel ebben a buszfázisban aktiválva van
nA: a jel ebben a buszfázisban nincs aktiválva
x : tetszőleges
S : a jel konstans marad (korábban aktiválva)

A következő táblázat az nBERR és az nHALT jelek kikapcsolásakor bekövetkező tényleges következményeket mutatja. Azt feltételezzük, hogy az nDTACK valamennyi esetben inaktív. Az nDTACK és az nBERR kikapcsolására csak akkor kerülhet sor, amikor a cím-strobe (nAS) már inaktív.

A megelőző esemény vége	**Vezérlőjel**	**Az ütem felfutó élére kapcsolva**			**Eredmény (következő ciklus)**
		N		**N+2**	
buszhiba	nBERR	*	vagy	*	buszhiba
	nHALT	*	vagy	*	kizárás következik
újraindítási ciklus	nBERR	*	vagy	*	nem megengedett következmény (többnyire a 0 vektorra fut)
	nHALT	*			
újraindítási ciklus	nBERR	*			újraindítási ciklus
	nHALT			*	
normál	nBERR	*			a következő ciklust meghosszabbíthatja
	nHALT	*	vagy	*	
normál	nBERR			*	ha egy újabb futna le, akkor egy buszhibakizárással befejezi
	nHALT	*	vagy		

3. A CSALÁD TOVÁBBI PROCESSZORAI

3.1 A 68008-as

A 68008-ast azért fejlesztették ki, hogy olcsóbb 68000-es rendszereket lehessen építeni. Adatbusza csak 8 bit szélességű. Ennek ellenére az utasításkészlete és az előállított programkódok teljesen kompatíbilisek a 68000-esével.

Annak érdekében, hogy az árat a lehető legkisebbre lehessen csökkenteni, a tokozáson a csatlakozók számát 48-ra csökkentették. Ily módon több csatlakozót takarítottak meg, mint amennyi a D8...D15 adatvonal csatlakozókkal felszabadult volna. Ezen az áron viszont a CPU további jeleiről is le kellett mondani.

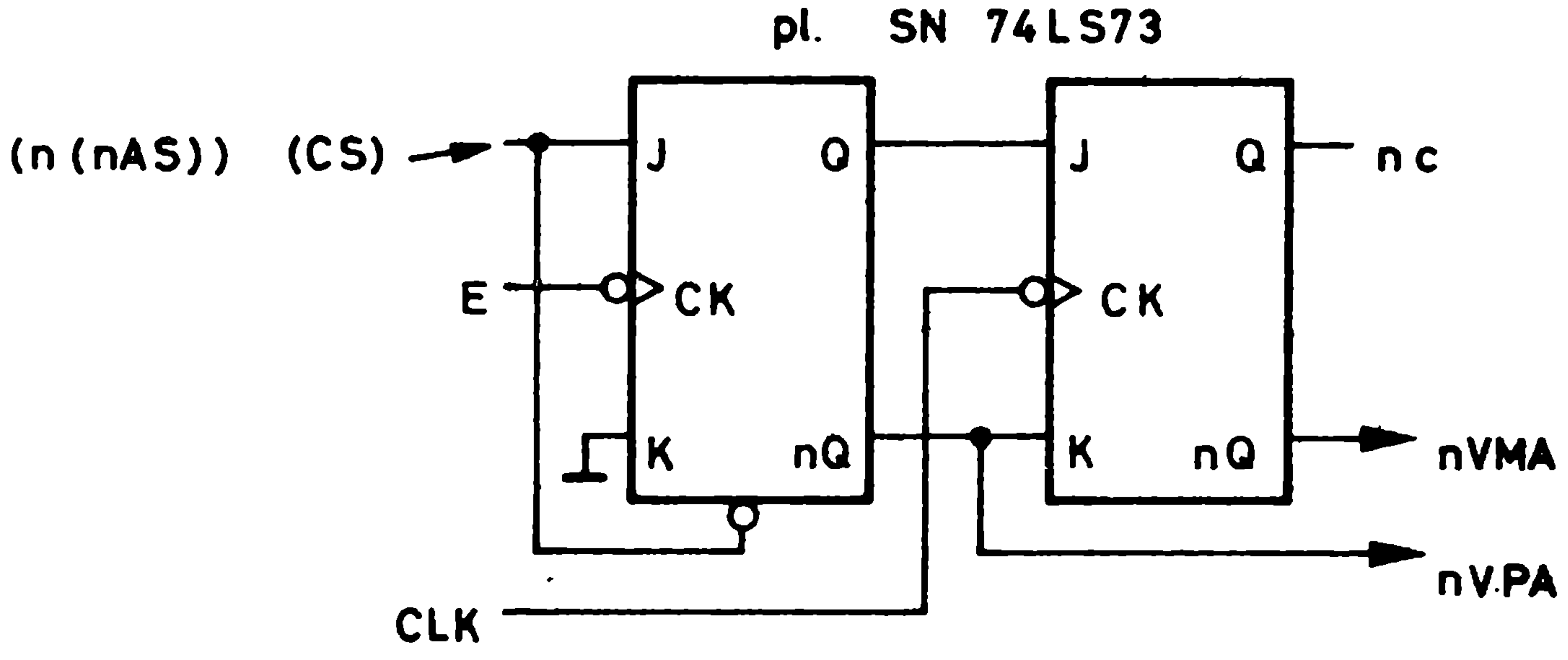
A címvezetékek számát 20-ra korlátozták. Mivel a tár – a 68000-estől eltérően – most valóban egyes byte-okból és byte-címekből áll, van A0 címvezeték is. A címvezetékek terjedelme tehát A0...A19. Így „csupán csak" egy megabytenyi címezhető terület áll rendelkezésre.

A megszakítások vezérlésére két csatlakozó alkalmas. Ezeket nIPL0/2-nek és nIPL1-nek jelöljük. Az nIPL0/2 csatlakozó belülről az nIPL0 és nIPL2 bemenetekkel van összekötve. Emiatt a 68008-nál három prioritásfokozat alkalmazható. A megszakítási bemenetek funkciója nem változott meg. Ha mindkét bemenet magas (HIGH), akkor nincs megszakítás. A prioritási fokozatokat az állapotregiszterben levő I0, I1 és I2 bitek maszkolják. A 7-es megszakítási fokozat (az nIPL0/2 vagy az nIPL1 alacsony) nem maszkolható (NMI = nem maszkolható megszakítás).

Az elérhető megszakítási fokozatok kódja a következő:

nIPL1	nIPL0/2	Megszakítási prioritás
L	L	7-es fokozat (NMI)
L	H	5-ös fokozat
H	L	2-es fokozat
H	H	0-ás fokozat, nincs megszakítás (nyugalmi állapot)

A 68008-nál a szinkron buszhozzáférés belső szinkronizálásáról és az nVMA jel előállításáról lemondtak. Egy viszonylag egyszerű kapcsolás alkalmazásával ez azonban kívülről is megoldható. A kapcsolásnak az (n(nAS))*(CS) jelölésű bemenetet magasra kell állítani akkor, amikor a címstrobe (nAS) aktív, és valamelyik szinkron üzemű egységnek a címe kiadásra került (CS, Chip-Select). A kapcsolás kimenetei megfelelnek a 68000-es kimeneteinek.



Az nVMA jel külső generálása

Mivel az adatbusz szélessége egy byte, egy adatengedélyezésre van szükség. A 68008-asnál az nDS (adatstrobe) jel helyettesíti a 68000-es nLDS és nUDS jelét.

nDS	R/nW	D0...D7
H	x	nincs érvényes adat
L	H	érvényes adatbitek (olvasás-hozzáférés)
L	L	érvényes adatbitek (íráshozzáférés)

A többi jel a 68008-asnál ugyanúgy működik, mint a 68000-esnél. A CPU buszhozzáférései és a buszvonalátadás technikája ugyancsak azonos. A 68000-es rendszerek rendszervezérlései is közvetlenül vonatkoztathatók a 68008-asra.

A 68008-asnak pontosan ugyanaz a regiszterkészlete, mint a 68000-esé, az utasításkészlet is azonos. A programok az egyik processzorról közvetlenül átvihetők a másikra.

A 68000-eshez hasonlóan a szó vagy a kettős-szó hozzáférések csak páros címekre irányulhatnak.

A tár byte-felépítésű. Ezekbe a 68008-as ugyanúgy helyezi el az adatokat, mint a 68000-es. A különbség csak az adatbusz szélességében van. Egy szó szélességű hozzáféréshez, pl. egy utasítás (16 bit) betöltéséhez a 68000-estől eltérően tehát két buszhozzáférésre van szükség.

A tárolóban az egyes adattípusok szervezése pontosan úgy történik, mint a 68000-esnél. A mindenkori magasabb helyiértékű szó a páros címen, az alacsonyabb helyiértékű szó a páratlan címen áll.

7 6 5 4 3 2 1 0		
0. byte	n	byte
1. byte	n+1	byte
2. byte	n+2	byte
3. byte	n+3	byte

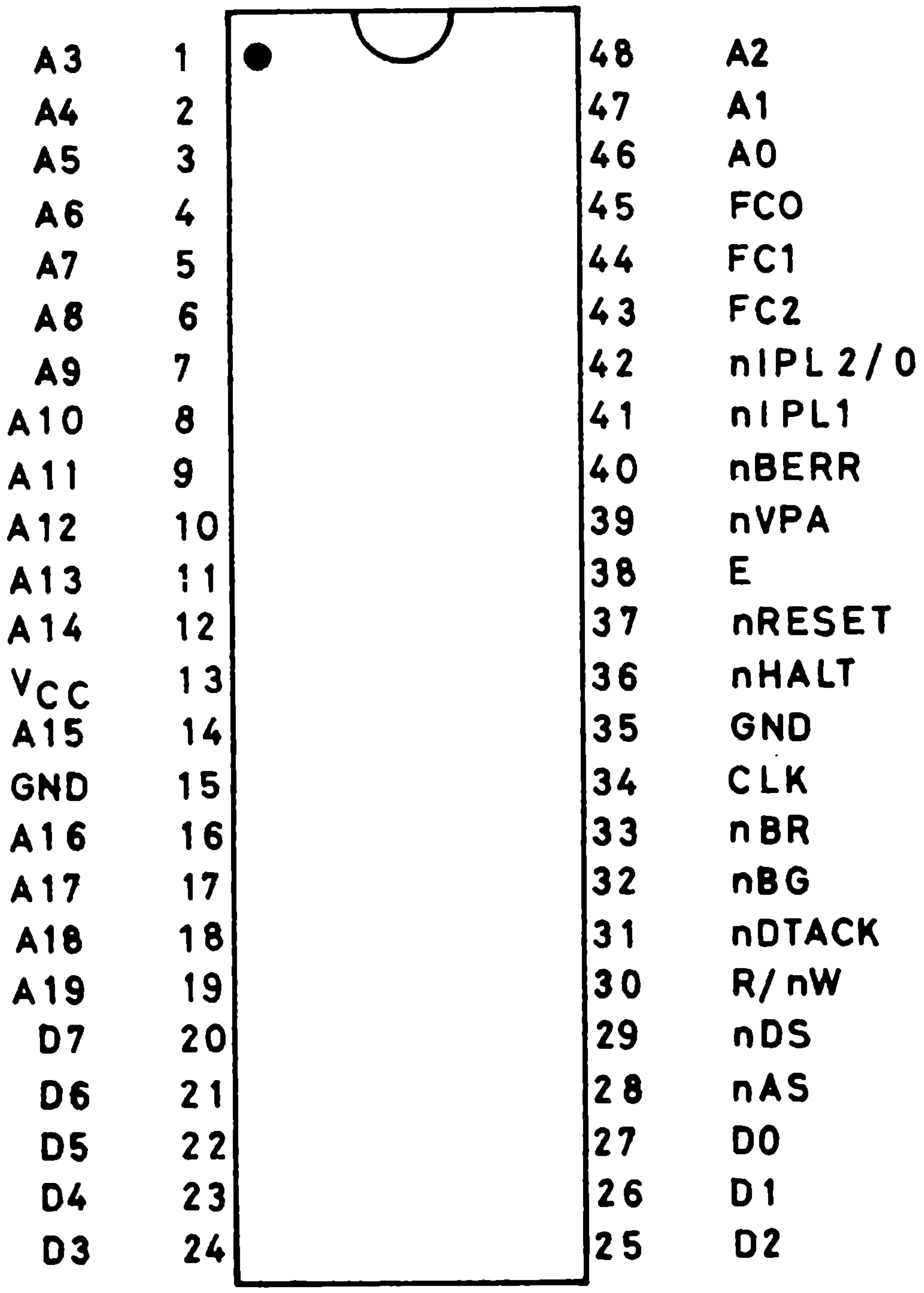
Egész számú adatok (byte)

7 6 5 4 3 2 1 0				
0. byte	magasabb helyiértékű byte		n	byte
		0. szó		
1. byte	alacsonyabb helyiértékű byte		n+1	byte
0. byte	magasabb helyiértékű byte		n+2	byte
		1. szó		
1. byte	alacsonyabb helyiértékű byte		n+3	byte

Egész számú adatok (szó)

7 6 5 4 3 2 1 0				
0. byte	magasabb helyiértékű		n	byte
1. byte	szó		n+1	byte
		0. kettős-szó		
2. byte	alacsonyabb helyiértékű		n+2	byte
3. byte	szó		n+3	byte
0. byte	magasabb helyiértékű		n+4	byte
1. byte	szó		n+5	byte
		1. kettős-szó		
2. byte	alacsonyabb helyiértékű		n+6	byte
3. byte	szó		n+7	byte

Egész számú adatok (kettős szó)


A3 1
A4 2
A5 3
A6 4
A7 5
A8 6
A9 7
A10 8
A11 9
A12 10
A13 11
A14 12
V_{CC} 13
A15 14
GND 15
A16 16
A17 17
A18 18
A19 19
D7 20
D6 21
D5 22
D4 23
D3 24
48 A2
47 A1
46 A0
45 FC0
44 FC1
43 FC2
42 nIPL2/0
41 nIPL1
40 nBERR
39 nVPA
38 E
37 nRESET
36 nHALT
35 GND
34 CLK
33 nBR
32 nBG
31 nDTACK
30 R/nW
29 nDS
28 nAS
27 D0
26 D1
25 D2

Az adatok tárbeli szervezése a 68008-asnál
(mindenkor az n címtől kezdődően)
A Dual-In-Line-tokozásban levő 68008-as mikroproceszor lábkiosztása

3.2 A 68010-es

A 68010-es a 68000-es közvetlen továbbfejlesztése. Ennek is 16 bites az adatbusza. A 68010-esnél lehetőség van arra, hogy egy regiszterrel (VBR = Vektor-Basis-Register) a tárban levő kizárási vektortáblázat helyét szabadon megválaszthassuk.

Újonnan került beépítésre egy MOVE utasítás (MOVES) is, amellyel tetszőleges adat- és programtartományokhoz lehet fordulni. Ez az utasítás a hozzáféréshez szükséges funkciókódokat két, egyenként három bit szélességű CPU-regiszterből (SFC, DFC) veszi elő. A regiszterek tartalma az „idegen" címtartományra irányuló buszhozzáférés során egyszerűen a funkciókód-kimenetekre (FC0...FC2) kerül.

Ahhoz, hogy ezeket az új regisztereket használni lehessen, további utasítások és utasítások variánsai kerültek beépítésre. Külön figyelmet szenteltek annak, hogy a strukturált programozást és a magas szintű nyelvek használatát támogassák.

Ezenkívül a 68010-esnél javították az utasítás-„prefetch"-t. Ennek eredményeként most 33 olyan utasítás létezik, amelyek a DBcc utasítással együtt úgynevezett Loop-módban (hurokmódban) működtethetők. A Loop-módban csak a feldolgozandó adatokhoz szükséges buszciklusok kerülnek végrehajtásra, a feldolgozandó utasítások a belső prefetch-regiszterekben, a hurkon belül maradnak. Ezáltal a tárterület-mozgatásokat és a keresési műveleteket lényegesen meg lehet gyorsítani. A felhasználónak a Loop-módot nem kell külön beállítania, a 68010-es automatikusan áttér erre.

A 68000-esnél az user-módban (felhasználói módban) még volt arra lehetőség, hogy a teljes állapotregiszter tartalmát elolvassuk. Ezt a 68010-es a user-módban nem teszi lehetővé. A MOVE from SR utasítás privilegizált. Viszont a 68010-esnél a MOVE from CCR utasítással userállapotban a Condition-Code-Register (CCR, az állapotregiszter user-byte-ja) tartalma, és így a kapcsolók (flagek) állapota beolvashatók.

A kizárások feldolgozását a 68010-esnél finomították. A 68010-es a busz- és címhibakizárások során több adatot helyez el a veremben, mint a 68000-es vagy a 68008-as. A 68010-esnél lehetőség van arra, hogy a kizárást kiváltó buszhozzáférést megismételjük. A felhasználó rendelkezésére állnak a szükséges információk a buszhozzáférés emulálásához.

Ez a két változtatás lehetővé teszi, hogy a 68010-essel virtuális processzor- és tárkoncepciókat lehessen megvalósítani.

A 68010-es tárterülete – a 68000-essel megegyezően – 16 Mbyte nagyságú. Ez a tárterület azonban csak igen ritkán telik meg egészen. A miniszámítógépek és a nagyszámítógépek már régóta un. virtuális tárkezelést használnak. Ilyenkor egy fizikailag meglevő tárat egy un. MMU (Memory-Management-Unit = tárkezelő egység) különböző tárterületekre helyez. Az MMU a program futása közben figyelemmel kíséri, hogy a hozzáférés egy nem jelenlevő tárterületre történt-e. Ha ez az eset áll fenn, akkor ez buszhibát vált ki. A buszhibarutinban a hiányzó tártartomány (a hiányzó „oldal") az egyik külső tárolóegységről betöltődik, ezt megelőzően azonban a valóságban is meglevő tár tartalma tárolásra kerül. Most történik meg a fizikailag meglevő tárnak arra a területre való helyezése, ahová a hozzáférés irányult. Ily módon viszonylag kis tárkapacitással nagy munkatárigényű programok is futtathatók. Az viszont fontos, hogy az utasítások, amelyeknél hiba lépne fel, megismételhetők vagy emulálhatók legyenek.

Az ilyen jellegű, virtuális tárkezelést a 68000-essel csak nehezen lehetett megvalósítani, mivel azok az információk, amelyek egy buszhibakizáráskor a supervisor-verembe kerültek, az utasítás megismétléséhez nem voltak mindig elegendők.

A 68010-essel viszont virtuális processzortechnika is használható. Ennek során a 68010-esen egy maszter operációs rendszert supervisor-módban futtatnak. Egy kifejlesztendő operációs rendszert user-módban futtatnak. Ha ez privilegizált, vagy nem meglevő utasítások végrehajtását kísérelné meg, akkor mindig egy kizárás kiváltása következik be. A kizárásrutin a maszter operációs rendszer része, a rutin a nem megengedett funkciókat emulálja vagy hibát jelez. Így az operációs rendszerek fejlesztése lényegesen leegyszerűsödik. Ezt az eljárást a 68000-esnél nem lehet alkalmazni, mert annál a felhasználói program is a teljes állapotregisztert olvasni tudta, tehát minden pillanatban azt is meg tudta állapítani, hogy az csak user-módban dolgozik.

A 68010-es regiszterkészlete

A regiszterkészlet a vektor-bázis-regiszterrel és az alternatív-funkciókód-regiszterrel kibővült. Az állapotregiszter változatlan maradt.

A vektor-bázis-regiszter mindig a kizárási vektortáblázat címét tartalmazza. Ez lehetővé teszi azt, hogy a tárban egyidejűleg különböző vektortáblázatot tartsunk és használjunk anélkül, hogy ezeket mindenkor át kellene másolni. Ezt a módszert használja a Z8000-es is.

A két alternatív-funkciókód-regiszter mindig azokat a funkciókódokat tartalmazza, amelyek a MOVES művelet során kerülnek az FC0...FC2 vezetékre. Az egyik regiszter akkor kerül alkalmazásra, amikor a MOVES utasítással adatokat egy „idegen" tárterületre kell átvinni, a másik akkor, amikor az „idegen" területről olvasni kell.

SFC Source-Function-Code	(= a forrás funkciókód) „idegen” tárolótérről való olvasáskor
DFC Destination-Function-Code	(= rendeltetési funkciókód) „idegen” tárterületekre való íráskor

A két regiszter még azt is lehetővé teszi, hogy olyan bitkombinációkat adjunk a funkciókód-vezetékekre, amit a processzor egyébként a normál olvasási vagy írási műveleteknél soha sem használ.

A MOVEC utasítás segítségével az új regiszterek olvashatók és írhatók, ezek során mindig kettős-szó műveletek kerülnek végrehajtásra. Az SFC és a DFC regiszterek olvasásakor a 3...31 bit értéke mindig nulla. Íráskor a legfelső 29 bit nem kerül figyelembevételre.

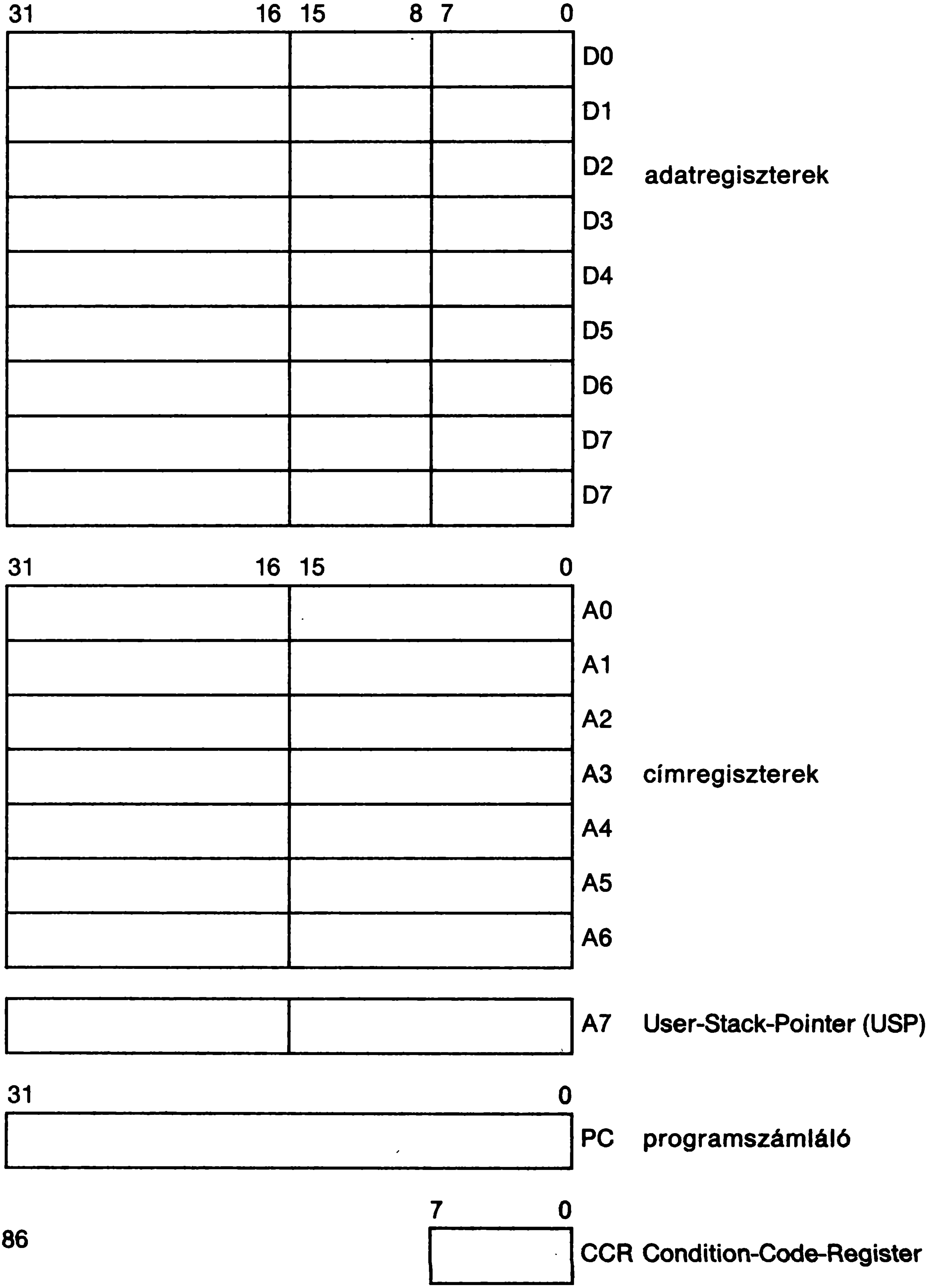
A többi regiszter működésmódján – az állapotregiszter kivételével – nem változtattak, így ezek a 68000-es regisztereivel teljesen kompatíbilisek.

Az adatok szervezése a 68010-es tárjában ugyanúgy történik, mint a 68000-esnél. A 68000-esen futó programok változtatás nélkül használhatók a 68010-esen. Csak azt kell figyelembe venni, hogy ezen utóbbinál kibővült az utasításkészlet, és hogy az állapotregiszter rendszerbyte-jának az olvasása privilegizált lett.

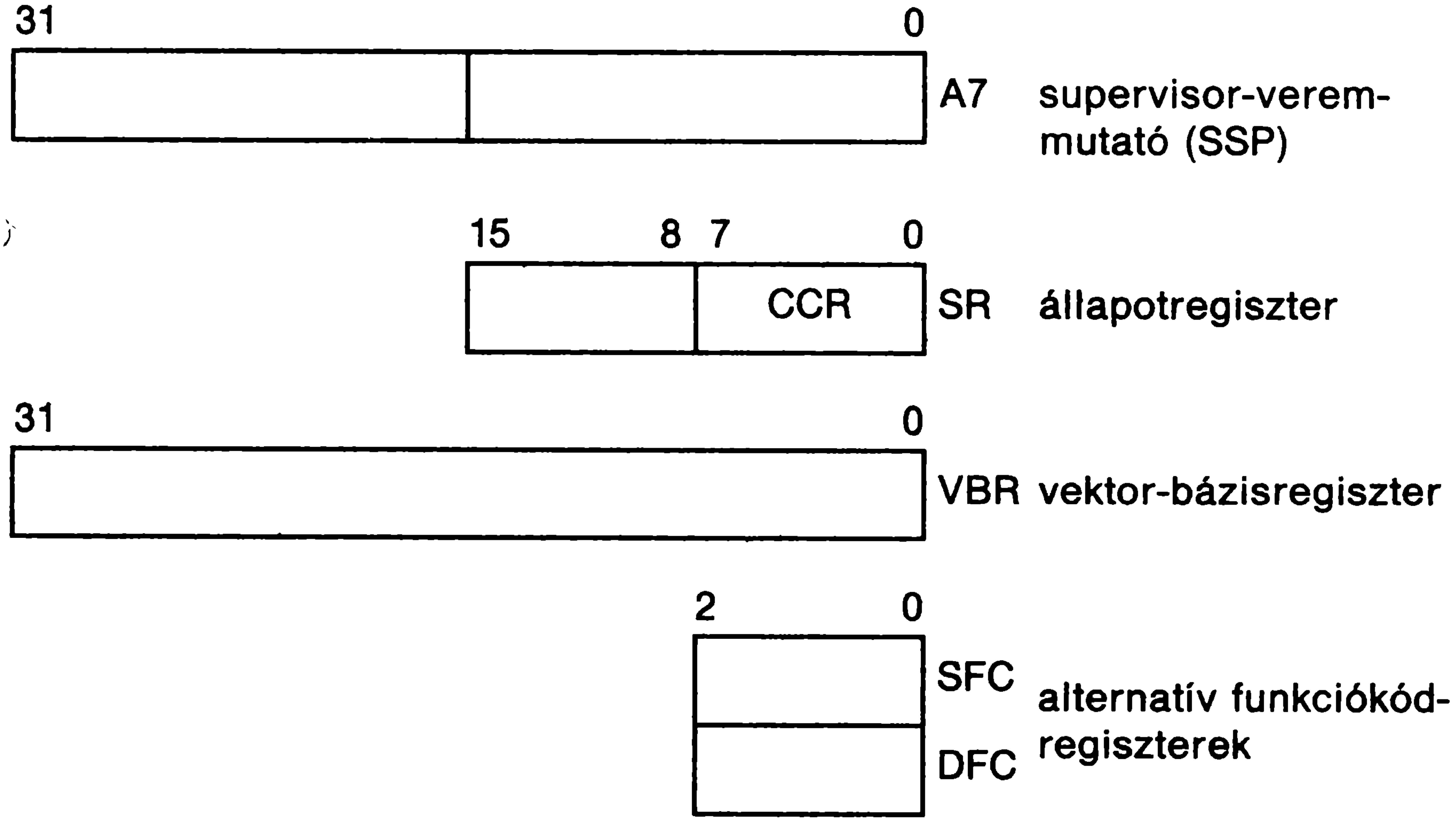
A következőkben néhány ábrán szemléltetjük a regiszterkészletet user- és supervisor-módban. A 68000-es állapotregisztere és a 68010-es állapotregisztere egymással teljesen megegyezik. Az állapotregiszter ábrája tehát mindkét processzorra (és természetesen a 68008-asra is) vonatkozik.

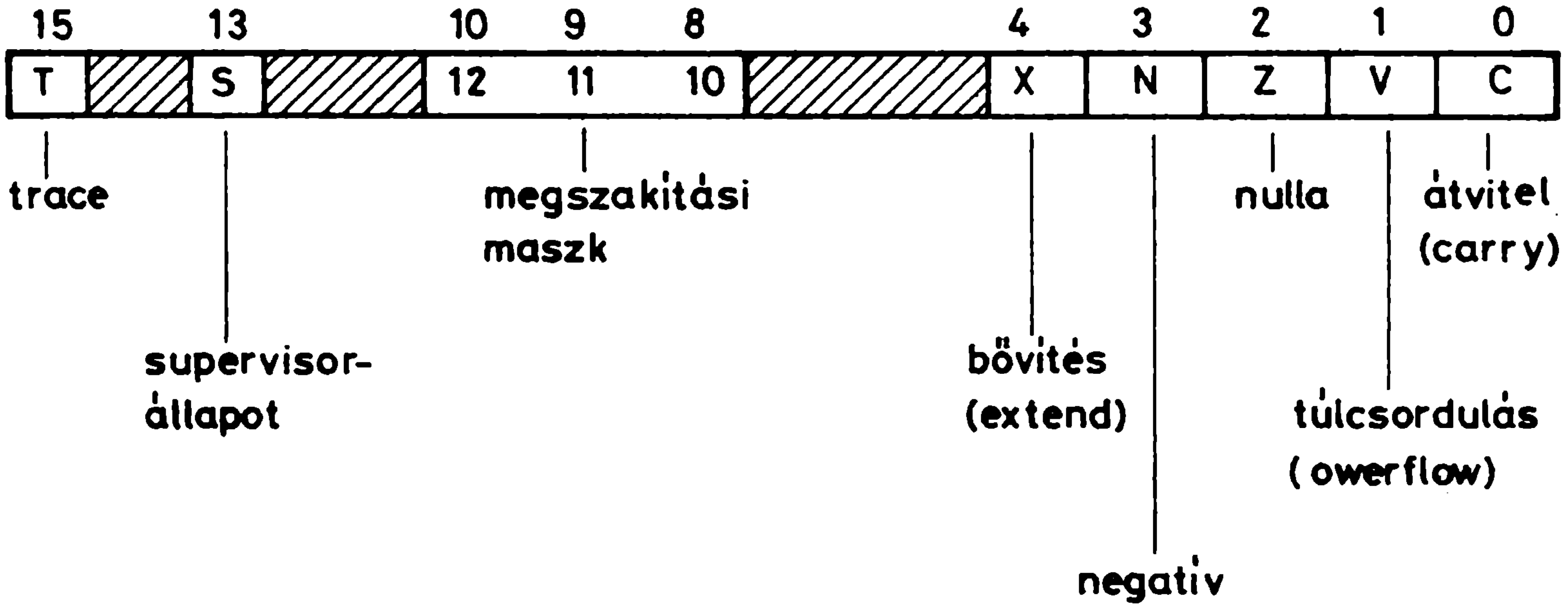
A 68010-es programozási modellje

Regiszterkészlet user-módban:


31
16
15
8
7
0
D0
D1
D2
D3
D4
D5
D6
D7
D7
adatregiszterek
31
16
15
0
A0
A1
A2
A3
A4
A5
A6
címregiszterek
A7
User-Stack-Pointer (USP)
31
0
PC
programszámláló
7
0
CCR
Condition-Code-Register

A kiegészítő regiszterek supervisor-módban:


31
0
A7
supervisor-verem-mutató (SSP)
15
8
7
0
CCR
SR
állapotregiszter
31
0
VBR
vektor-bázisregiszter
2
0
SFC
DFC
alternatív funkciókód-regiszterek


15
13
10
9
8
4
3
2
1
0
T
S
12
11
10
X
N
Z
V
C
trace
supervisor-állapot
megszakítási maszk
bővítés (extend)
negatív
nulla
túlcsordulás (owerflow)
átvitel (carry)

A 68010-es állapotregisztere

A kizárások (exceptions) kezelése a 68010-esnél

A 68010-es a kizárásokat hasonlóan kezeli, mint a 68000-es. Van azonban néhány változtatás, hogy a virtuális processzor- és tárkoncepciókat jobban lehessen támogatni. Ehhez a 68010-es kizárásai során más, és több információ kerül a verembe. Ezenkívül lehetőséget teremtettek arra, hogy olyan töréspontokat (breakpoints) lehessen elhelyezni, amelyek kiváltásukkor jeleket adnak a külső hardver részére.

A 68000-es egyik hátránya az volt, hogy a kizárási vektortáblázat fixen a tár elején volt. Ha ennél alternatív vektortáblázatot akartunk volna használni, akkor ehhez viszonylag hosszú számítási időre lett volna szükség, mert 1024 byte-ot (ill. 256 kettős-szót) kellett volna mozgatni. Jobb az a koncepció, amelyet más processzorok, így pl. a Zilog Z8000-ese használ. A processzor egy regisztere megadja a vektortáblázat elejének a helyét. Erre a célra hozták létre a 68010-esnél a VBR-t (vektor-bázis-regiszter). Ez 32 bit szélességű. A kizárási vektortáblázatot így a tár tetszőleges helyére el lehet helyezni. A regiszter mindig a vektortáblázat elejére (az alacsonyabb helyiértékű címre), azaz az első RESET vektor helyére mutat (a veremmutató kezdeti értéke).

A VBR regiszterbe írni, ill. onnan olvasni a MOVEC utasítással lehet. A VBR regiszterhez valamennyi hozzáférés kettős-szó szélességű. Ezért, hogy a rendszer meghatározott állapotból indulhasson, a VBR regisztert nullára kell állítani (törölni).

Egy kizárási művelet lefolyása a 68010-esnél kis különbségektől eltekintve megfelel a 68000-esének vagy a 68008-asának. A kizárásnak az RTE (ReTurn from Exception) utasítással való lezárásánál változások kerülnek végrehajtásra. Ennél az utasításnál a 68010-es először ismét beolvassa a veremben tárolt belső regisztert. Ennek során több ellenőrzést végez el.

A 68010-esnél van egy új kizárás:

FORMÁTUMHIBA

a kizárás száma: $OE
a kizárás vektora: $038...03C

A 68010-es egy kizárás megkezdésekor több információt helyez a verembe, mint a 68000-es vagy a 68008-as. Ezeket az információkat az RTE utasítás végrehajtásakor újra elolvassa. Ennek során a 68010-es felülvizsgálja az adatokat. Ha hibát észlel, akkor váltja ki ezt a kizárást.

Ez a kizárás a kettes prioritási csoportba tartozik.

Az *ILLEGÁLIS UTASÍTÁS* nevű kizárás (száma: 4, a vektor címe : $010...013) funkciója valamelyest megváltozott. Nyolc, a 68000-esnél is meglévő illegális utasításkódhoz a 68010-esnél egy további, új funkciót rendeltek hozzá.

A $4848...$484F utasításokról van szó. Ha megkíséreljük ezen utasítások valamelyikének végrehajtását, a 68010-es egy töréspont-ciklust végez el, mielőtt még az „illegális utasítás" kizárási rutint meghívná.

Egy töréspont-ciklus során a 68010-es az FC0...FC2 vezetékeket $111-re állítja. Ehhez valamennyi címvezetéket aktiválja, és mindegyiket alacsony szintre állítja. Ebben a ciklusban a processzor sem nem fogad, sem nem küld adatokat. Az R/nW vezeték a ciklus teljes ideje alatt magas szintre van állítva. A töréspont-ciklust egy nDTACK, egy nBERR vagy egy nVPA jel fejezi be. A töréspont-ciklus befejeződésekor egy normál kizárás hajtódik végre.

Az ilyen utasítások valamelyikének a kiadásával a külső hardver számára közvetlenül jelezhető a töréspont fellépése.

A 68000-esnél az FC0...FC2 vezetékeken a $111 bitkombináció egyértelműen megszakítás-nyugtázást jelentett. Ezzel szemben ez a bitkombináció a 68010-esnél (és a 68020-asnál) egy CPU-space-ciklust jelez. Ha az A4...A23 címvezetékek alacsonyak, akkor töréspont-ciklus áll fenn, ha magasak, akkor a 68010-es megszakítást nyugtáz. A 68000-esre épülő rendszerek fejlesztésénél is célszerű ellenőrizni az adatvezetékeket, hogy a későbbiekben 68010-est is lehessen alkalmazni.

A 68010-esnél a PRIVILÉGIUM MEGSÉRTÉSE kizárás az állapotregiszter egészének user-módban való olvasásakor is kiváltásra kerül. A 68010-esnél ezért létezik a MOVE from CCR utasítás, hogy a kapcsolók helyzetét (flag-eket) a user-módban is le lehessen olvasni. Erre a változtatásra azért került sor, hogy a virtuális processzortechnika és a processzoremulációk is megvalósíthatók legyenek.

A 68010-esnél a kizárás során a többi adat a supervisor-verembe kerül. A negyedik, utolsó szó tartalmazza, a legalsó 12 bitjében, a vektor-offszetet (azaz a két bittel eltolt kizárási számot). Ezáltal általánosan használható kizárási rutinok írhatók, amelyek először önmagukban megvizsgálják, hogy melyik kizárás esete forog fenn. Ennek a szónak a négy legmagasabb bitje tartalmazza az ún. formátumkódot:

Formátumkód	**Leírása**
0000	rövid veremformátum (négy szó), 1-es vagy 2-es prioritású kizárásoknál
1000	hosszú veremformátum (29 szó), cím- vagy buszhibakizárásoknál

A busz- vagy címhibáknál a 68010-es több adatot helyez a verembe, mint a 68000-es, hogy virtuális tárkezelést lehessen megvalósítani.

A 68010-es a busz- és címhibák előfordulásakor gyakorlatilag teljes belső állapotát a verembe tölti. Így a 68010-es RTE utasításának a végrehajtásakor lehetőség van a hibás utasítás megismétlésére (a 68000-es RTE utasításától eltérően).

A 68010-es prefetch-mechanizmusa miatt a tárolt programszámláló nem okvetlenül arra a címre mutat, amihez a processzor akkor fordult, amikor a hiba bekövetkezett, ezért még a hiba bekövetkezésekor fennállt címet is tárolni kell. Ezt a címet az operációs rendszer például arra használhatja, hogy megállapítsa, egy külső tárolóegységről melyik virtuális címet kell utántölteni. Ehhez még tárolni kell a bemeneti és kimeneti adatpuffert és az utasítás bemeneti pufferét.

Az a felhasználóra van bízva, hogy azt az utasítást, amelynek végrehajtása során a hiba fellépett, a processzorral megismételtesse, vagy pedig a hibás buszhozzáférést emulálja. Gyakran kedvezőbb az utasítás szoftveroldali befejezése. Az ugyancsak a supervisor-veremben tárolt kiegészítő állapotszóban lehet a processzor tudomására hozni, hogy a hibás hozzáférést ismételje meg, vagy pedig a verem tartalmát használja. E célra építették be a kiegészítő állapotszóba az RR- (Re-Run) bitet.

A 68010-es a kiegészítő állapotszóban adja meg azt is, hogy írási vagy olvasási hozzáférés történt, hogy ez egy byte vagy szó szélességű hozzáférés volt, és adott esetben, hogy a hozzáférés a magasabb vagy az alacsonyabb helyiértékű byte-ra történt.

Ezen kívül a processzor magasra állítja az RM-bitet (RM = Read-Modify), ha a hibát egy olvasás-módosítás hozzáférés váltotta ki. Egy olvasás-módosítás-írás ciklusnál a 68010-es abból indul ki, hogy a teljes ciklus szoftver útján befejeződött. Ezért az íráshozzáférést is át kell venni a kizárási rutinból, ha a hiba az olvasás-hozzáférés közben lépett fel.

A kiegészítő állapotszó felépítése a következő:

Bit	Jelölés és leírás
0...2	FC0...FC2 Ezek a bitek tartalmazzák azt a funkciókódot, ami a hibás ciklus alatt került felhasználásra.
3...7	Nem használt, olvasáskor nulla.
8	RW Ez a bit tartalmazza az R/nW-vezeték állapotát a hibás buszciklus közben.
9	BY Ez a bit akkor van magasra állítva, ha a hibás ciklus egy byte szélességű hozzáférés volt.
10	HB Ez a bit akkor van magasra állítva, ha egy byte szélességű hozzáférésnél a szó magasabb helyiértékű byte-jára irányult a hozzáférés.
11	RM Ez a bit akkor van magasra állítva, ha a hiba egy olvasás-módosítás-írás ciklus (a második típusú) során lépett fel.
12	DF Adatbemeneti pufferhez való hozzáférés.
13	IF Utasításbemeneti pufferhez való hozzáférés. Ezek a bitek akkor vannak magasra állítva, ha egy olvasás-hozzáférés során az adatokat a mindenkori pufferbe kell tölteni.
14	Nem használt, olvasáskor nulla.
15	RR Re-Run kapcsoló (Re-Run-Flag). Ha az a bit magasra van állítva, akkor a processzor az előző, hibás ciklust nem ismétli meg, hanem a veremben levő adatokat használja.

A supervisor-verem szervezése kizárásoknál (68010)

	15	0	
	0000	vektor-offszet	
	programszámláló (alacsonyabb helyiértékű szó)		magasabb
	programszámláló (magasabb helyiértékű szó)		címek
SP →	állapotregiszter		

Szervezés a buszhiba- és címhibakizárások esetén

	15	0	
	belső információk (16 szó)		
	utasításbemeneti puffer		
	nem használt (foglalt)		
	adatbemeneti puffer		
	nem használt (foglalt)		
	adatkimeneti puffer		
	nem használt (foglalt)		
	hiba címe (alacsonyabb helyiértékű szó)		
	hiba címe (magasabb helyiértékű szó)		
	speciális állapotszó		
	1000	vektor-offszet	magasabb
	programszámláló (alacsonyabb helyiértékű szó)		címek
	programszámláló (magasabb helyiértékű szó)		
SP →	állapotregiszter		

Az RTE utasítás a 68010-esnél

Az RTE utasítás (ReTurn from Exception = visszatérés a kizárási állapotból) végrehajtásakor a 68010-es több vizsgálatot végez el. Az RTE utasítás végrehajtásának megkezdésekor a processzor mindenkor újra beolvassa az állapotregisztert, a programszámlálót és a negyedik szót a formátumkóddal a veremből.

Először a veremformátumot vizsgálja meg. Ha a 68010-es rövid veremformátumot észlel, akkor a program végrehajtását a programszámláló régi tartalmától kezdődően folytatja. További vizsgálatokra kerül sor akkor, ha hosszú veremformátumot jelző kódot talál. Minden más esetben a formátumhiba-kizárás kerül kiváltásra.

Ha hosszú veremformátumot talált, akkor a 68010-es egy további szót olvas be a veremből (az SP+26 címről). Ebben a szóban a CPU valamelyik verziójának a számát tárolja. Ha ez nem egyezik meg a CPU számával, akkor ugyancsak egy formátumhiba-kizárás kerül kiváltásra. Ily módon lehet a különböző verziójú, több 68010-es processzorból felépülő (többprocesszoros) rendszerekben azokat a problémákat elkerülni, hogy a különböző verziók a belső információkat adott körülmények között különböző módon tárolják.

Ezenkívül hosszú veremformátum esetén a processzor beolvassa még a kizárás veremtartományának a legutolsó szavát (az SP+56 címről). Ha ilyenkor a külső hardver buszhibát jelez, akkor egy normál buszhibakizárás kerül kiváltásra. Ezután a 68010-es valamennyi, korábban tárolásra került szót beolvassa. Ha ennek során buszhiba lép fel, akkor ezt a 68010-es kettős buszhibának minősíti, és átmegy állj állapotba.

A 68010-es csatlakozói és jelei

A 68010-es lábkiosztást illetően a 68000-essel kompatíbilis. A két processzor tehát egymással közvetlenül felcserélhető. Így a 68000-es lábkiosztását bemutató ábra a 68010-esre is ugyanaz.

A 68010-es pontosan azokat a buszciklusokat hajtja végre, mint a 68000-es, az vonatkozik a megszakításokra és a buszvonalátadásra (Bus-Arbitration) is.

A funkciókód-vezetékek alkalmazása a 68010-esnél némiképp változott. Az FC0...FC2 vezetékeken a $111 kombináció a 68010-esnél már nem jelenti egyértelműen a CPU megszakítás nyugtázását. A $111 kombináció a 68010-esnél egy CPU-space-ciklusra utal. Amennyiben ez egy megszakítás nyugtázása, akkor az A4...A23 címvezetékek magasra állítódnak (A1...A3 a nyugtázott megszakítás prioritási fokozatát tartalmazza). Ezzel szemben egy töréspont-ciklus esetén valamennyi címvezeték alacsony szintre állítódik.

Természetesen nincs szükség valamennyi címvezeték vizsgálatára. Ahhoz azonban, hogy a CPU későbbi verzióval és variánsaival kompatíbilisek maradjunk, minimum az A16...A19 címvezetékeket ellenőrizni kell.

Egy töréspont-ciklusnál a funkciókód- és a címvezetékek az S0 kezdetén, ill. végén magas szintre állítódnak. Az S2 felfutó élére az nAS, nLDS és nUDS aktivizálódik. Az R/nW a teljes töréspont-ciklus alatt magas állapotú. Az S4 lefutó élétől kezdődően a 68010 a töréspont-ciklusnak az nDTACK, nVPA vagy az nBERR jellel való befejezésére vár. A legtöbb alkalmazásban legegyszerűbben az nDTACK jellel való befejezés valósítható meg.

A hurokmód (Loop-mód)

A 68010-esnek kettős-szó szélességű prefetch-technikája és egy szó szélességű utasításregisztere van. Ez tette lehetővé a hurokmód implementálását.

Amikor a 68000-esnél egy DBcc utasítás közvetlenül a DBcc utasítást megelőző utasításra ágazik el, akkor a 68010-es bizonyos körülmények között beléphet hurokmódba. Ha ugyanis a program e hurkon belüli lefolyásában a prefetch-regiszterek és az utasításregiszter elegendő szélességű e hurok valamennyi szavához, akkor a 68010-es nem hajt végre további utasításbetöltő ciklusokat. Ehelyett az utasításokat a belső regiszterekben tartja. Ezt az állapotot nevezték el hurokmódnak.

A hurokmódra nincs valamennyi utasításnál lehetőség. Csak akkor indítható, ha a DBcc utasítás mínusz néggyel vagy mínusz kettővel ágazik el, és az utasítás általában engedélyezett a hurokmód számára. A következő összeállítás azokat az utasításokat sorolja fel, amelyeknél egy DBcc utasítás előtt a hurokmód lehetséges:

Utasítások	**Lehetséges címzésmódok**
MOVE	(Ay) to (Ax), (Ay) to (Ax)+, (Ay) to −(Ax)
	(Ay)+ to (Ax), (Ay)+ to (Ax)+, (Ay)+ to −(Ax)
	−(Ay) to (Ax), −(Ay) to (Ax)+, −(Ay) to −(Ax)
	Ry to (Ax), Ry to (Ax)+
ADD	(Ay) to Dx
AND	(Ay)+ to Dx
CMP	−(Ay) to Dx
OR	
SUB	
ADDA	(Ay) to Ax
CMPA	−(Ay) to Ax
SUBA	(Ay)+ to Ax
ADD	Dx to (Ay)
AND	Dx to (Ay)+
EOR	Dx to −(Ay)
OR	
SUB	
ABCD	−(Ay) to −(Ax)
ADDX	
SBCD	
SUBX	
CMP	(Ay)+ to (Ax)+
CLR	(Ay)
NEG	(Ay)+
NEGX	−(Ay)
NOT	
TST	
NBCD	

Utasítások	Lehetséges címzésmódok
ASL	(Ay) #1-gyel
ASR	(Ay)+ #1-gyel
LSL	−(Ay) #1-gyel
LSR	
ROL, ROR	
ROLX, RORX	

A hurok-módból megszakításokkal vagy kizárásokkal lehet kilépni. Természetesen egy DBcc is befejezi. A megszakítások csak a DBcc utasítás végrehajtását követően megengedettek, a hurok más utasítása után nem.

A 68012-es

A 68000-es családban van még egy további, szintén 16 bit szélességű adatbusszal rendelkező processzor.

A 68012 felépítését illetően megegyezik a 68010-essel. Ezzel szemben viszont a címtartományának 2 MByte a kapacitása. Ehhez további hét címvezeték tartozik. Ezek az A24...A29 és az A31 vezetékek. Az A30-as vonal nem áll rendelkezésre, a 68012-es címtartománya tehát „fel van darabolva”. A belső, A30-as vezeték állapotától függetlenül a címek a megfelelő vezetékekre kerülnek. Így a címtartományban tükröződések léphetnek fel.

A 68012-esnek – a 68020-ashoz hasonlóan – van egy nRMC vezetéke (Read-Modify-Write-Cycle, olvasás-módosítás-írás ciklus). Ezen a vezetéken keresztül jelzi a processzor az olvasás-módosítás-írás ciklusokat. A 68000-esnél (és a 68008-asnál, 68010-esnél) az olvasás-módosítás-írás ciklusokat arról lehet felismerni, hogy az nAS a teljes ciklus alatt aktív marad. A 68012-esnél (és a 68020-asnál) az nAS az olvasási és az írási hozzáférés között egy olvasás-módosítás-írás ciklusban kikapcsolásra kerül.

Ezt leszámítva a 68012 és a 68010-es teljesen azonos. Ebből következően a 68012-es csak azoknál a rendszereknél lehet érdekes, ahol igen nagy a tár kapacitása, vagy amelyeknél szoftver útján a programok számára igen nagy virtuális tárterületet bocsátanak rendelkezésre.

3.3 A 68020-as

A 68020-as a 68000-es család első 32 bites mikroprocesszora. A 68020-as számos részében továbbfejlesztésre került. Ennek ellenére ez a programokat illetően kompatíbilis a család többi tagjával. A kompatibilitást illetően bizonyos korlátozások vannak azokra a rutinokra vonatkozóan, amelyek a kizárásokat dolgozzák fel. Felhasználói oldalról nézve azonban a 68020-as valamennyi, más processzorra írt programot változtatás nélkül át tud venni.

A 68020-as sok területen már annyira kibővült, hogy már szinte alig illik bele a 68000-es családba. A 68020-asnak akkora a teljesítménye, amelyet eddig jóformán csak a miniszámítógépekkel értek el.

A 68020-as rendszer lényegesen drágább, mint a család más processzoraira épülő rendszerek. Ez egyrészt magának a processzornak a magasabb árából, másrészt a 68020-as hardverigényéből adódik. Ezzel szemben áll a 68020-as igen széles elterjedtsége. A 68020-as részletes műszaki ismertetése minden további nélkül egy önálló könyvet igényelne. Ezért úgy döntöttünk, hogy a 68020-assal nem foglalkozunk olyan részletesen, mint a többi processzorral.

Lényegében ismertetjük a 68020-as elvi működésmódját, de a továbbiakat illetően messzemenően a progamozói-felhasználói vonalra korlátozódunk. A 68020-as utasításait teljes egészében tartalmazza az utasításkészletet ismertető fejezet. A pontos műszaki részletkérdések csak kevesek részére lehetnek érdekesek.

Amint már említettük, a 68020-asnak az adatbusza 32 bit szélességű. A hozzáférések aszinkron módon történnek (a 68020-asba szinkron buszhozzáférések már nincsenek beépítve). Arra azért van lehetőség, hogy a buszon 8 vagy 16 bites perifériaegységeket is működtetni tudjon anélkül, hogy ezt a programozás során figyelembe kelljen venni. Egy hozzáférés során a perifériákkal mindössze annyit kell közölni, hogy hány bitet vett át (ill. adott át). Az esetlegesen meglevő „maradékot" a 68020-as további buszhozzáférésekben dolgozza fel. Ezt az eljárást dinamikus buszhozzáférésnek nevezik. A többprocesszoros rendszereknél ez az eljárás jelentős előnyöket kínál.

A 68020-as címbusza 30 (+2) bit szélességű. Ezzel 4 gigabyte (ill. 2 gigaszó, giga-kettős-szó) címezhető meg. A 68020-asnál sem használják egyik buszvezetéket sem multiplex üzemmódban. A 68020-as egy buszhozzáféréséhez minimum 3 rendszerütemciklusra van szükség (a 68000-es 4 ütemciklusához képest).

A 68020-as prefetch-technikáját tovább javították, ezenkívül a chipbe be van építve az utasítások számára egy cache-tároló (cache = rejtekhely).

A 68020-assal is megvalósíthatók virtuális tároló- és processzorkoncepciók. Ezenkívül közvetlen lehetőség van arra, hogy társprocesszorokkal (koprocesszorokkal) kommunikáljon.

A 68020-asba további címzésmódokat építettek be. Ehhez új adattípusokat alakítottak ki. A társprocesszor-koncepció azt is lehetővé teszi, hogy az alkalmazástól függően további adattípusokat lehessen bevezetni. Itt szinte valamennyi utasítás 32 bit szélességű operandussal dolgozik.

A 68881-es lebegőpontos társprocesszorral összekapcsolva egyszerűen feldolgozhatók a tizedestörtek (az IEEE szabványnak megfelelően).

A 68020 regisztereinek szerkezete

A 68020-as regiszterkészletét kibővítették, hogy az új feladatokat el tudja látni.

A 68020-asnak három veremmutató-regisztere van (USP = User-Stack-Pointer = felhasználói veremmutató, ISP = Interrupt-Stack-Pointer = megszakítási veremmutató, MSP = Master-Stack-Pointer = maszter veremmutató). A megszakítási és a maszter veremmutató (ISP és MSP) a többi típusnál meglevő supervisor-veremmutató (SSP) szerepét veszi át. Azt, hogy a supervisor-módban melyik veremmutató legyen aktív, az állapotregiszterben levő maszter/megszakítási bit határozza meg (ha a bit 0, akkor az ISP aktív, ha a bit értéke 1, akkor az MSP aktív). Az átkapcsolás csak supervisor-módban lehetséges. Ha ezt a bitet nem változtatjuk, akkor a 68020-as a veremmutató használata esetén is kompatíbilis marad a család többi processzorával.

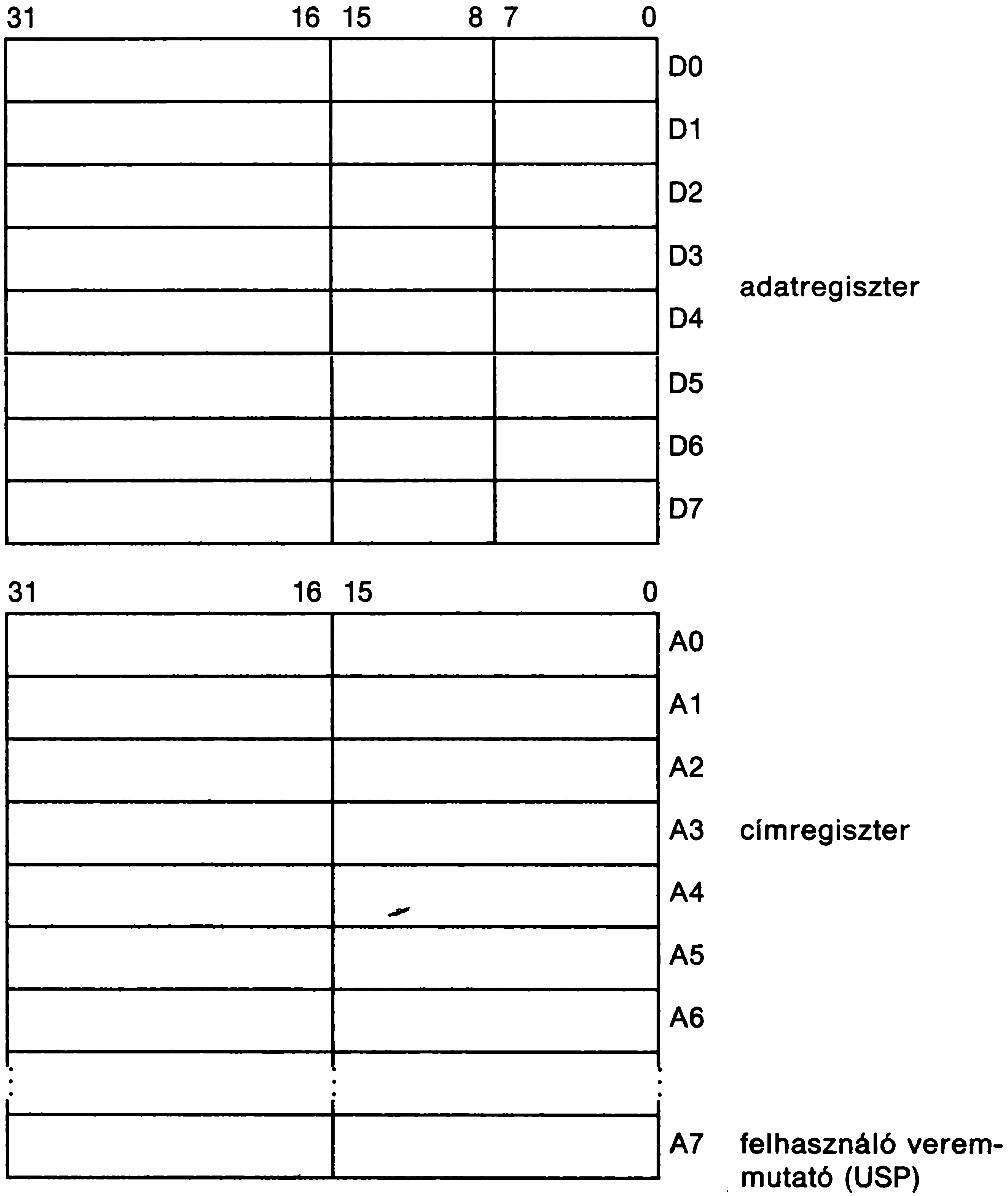
A vektor-bázis-regiszter itt is (vö. 68010-essel) a tárolóban levő vektortáblázat elejére mutat. Az SFC és DFC regisztereknek is ugyanaz a feladatuk, mint a 68010-esnél.

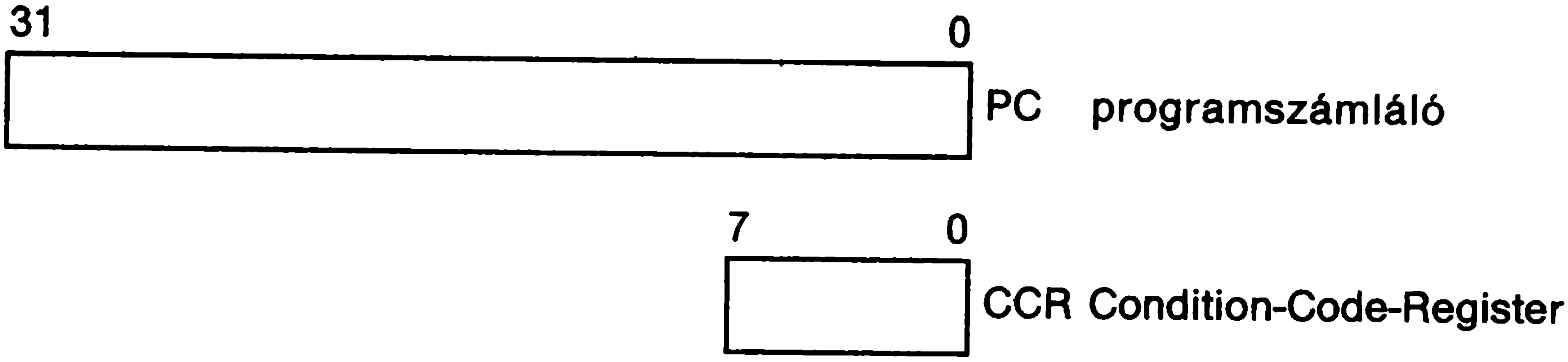
A CACR (CAche ContRol) és a CAAR (CAche-AddRess) regiszterek a belső cache-tároló vezérlésére szolgálnak.

A felhasználói programozói modell valamennyi regisztere (D0...D7, A0...A6, A7, CCR) teljes mértékben megfelel a családban levő más processzorok megfelelő regisztereinek.

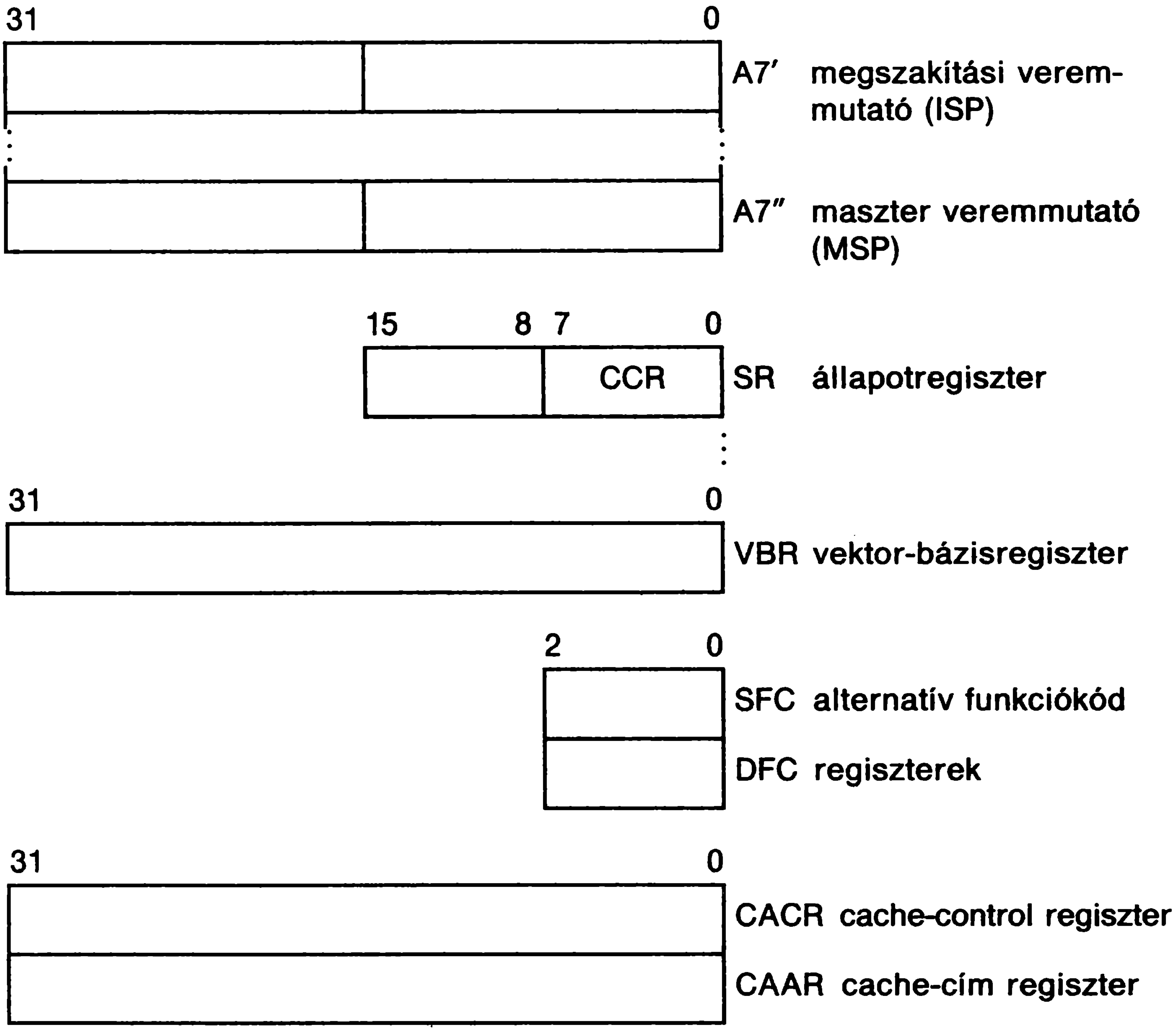
A 68020-as programozási modellje

Regiszterkészlet user-módban:

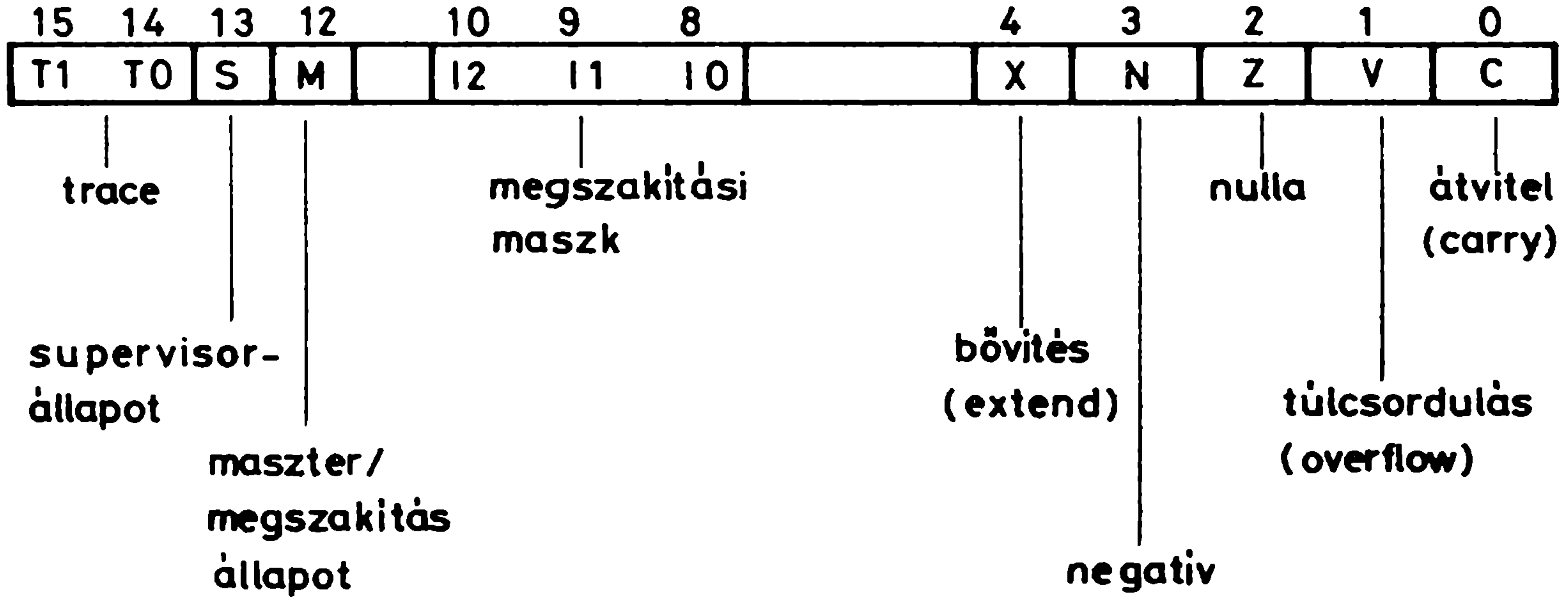

31
16
15
8
7
0
D0
D1
D2
D3
D4
D5
D6
D7
adatregiszter
31
16
15
0
A0
A1
A2
A3
A4
A5
A6
A7
címregiszter
felhasználó verem-mutató (USP)


PC programszámláló
CCR Condition-Code-Register

További regiszterek supervisor-módban


31
0
A7′ megszakítási verem-mutató (ISP)
A7″ maszter veremmutató (MSP)
15
8
7
0
CCR
SR állapotregiszter
31
0
VBR vektor-bázisregiszter
2
0
SFC alternatív funkciókód
DFC regiszterek
31
0
CACR cache-control regiszter
CAAR cache-cím regiszter

A 68020-as állapotregisztere



A tároló szervezése és a hozzáférési lehetőségek

A 68020-as tárolójának szervezése megfelel a 68000-esének. A 68020-as tárolója is byte-szervezésű. Mivel azonban az adatbusz szélessége 32 bit, mindig 4 byte „áll egymás mellett”. Ez azt jelenti, hogy a 68020-as tárolójában fizikailag kettős-szók (longwords) vannak, ezért a cím az egyik kettős-szótól a következő kettős-szóig (az egyik tárolóhelytől a következő tárolóhelyig) mindig néggyel nő.

A 68020-as azonban azt is lehetővé teszi, hogy szavas vagy byte-os perifériákkal vagy tárolókkal dolgozzon együtt. Ezt az eljárást dinamikus hozzáférésnek nevezik. Egy buszciklus lezárásaként a megszólított periféria kódolt formában mindig megküldi a kapu (port) szélességét (a címzésnél az adatbusz szélessége). Ezeken a kis adatbusz szélességű területeken a címek szervezése ugyanúgy történik, mint a 68000-esnél, ill. a 68008-asnál.

A 68020-as más adattípusokkal is dolgozik. A 68020-asba a tömörített BCD (binárisan kódolt decimális) adatok mellé – amelyek a család más processzorainál is léteznek – beépítették még a nem tömörített BCD adatokat is. Ennél az adattípusnál egy BCD-szám egy byte-nak csak az alsó négy bitjét foglalja el. A byte felső négy bitjét a felhasználó határozhatja meg.

A teljes, 32 bit szélességű szorzási és osztási utasítások beépítése miatt még 64 bit szélességű adatokat is definiálni kellett. Ezeket az adatokat négyes-szó szélességű adatoknak nevezzük (Quadruple-words vagy röviden Quadwords-adatok). Ezekre explicit módon vonatkoztatható utasítás nincs (a MOVEM utasításon kívül). Ezeknek az adatoknak a tárolása ugyanúgy végezhető, mint az egész számú byte, szó vagy akár kettős-szó szélességű adatoké. A négyes-szó szélességű adatok tárolása mindig két adatregiszterben történik, a regiszterek párosítására vagy sorrendjére vonatkozóan nincs semmiféle szabály vagy megszorítás.

A 68020-asnál kiegészítésként ezekhez az adatokhoz először kerültek bevezetésre a bitmezőadatok is. Bitmező alatt egy olyan meghatározott számú bitből álló sort értünk, amely egy meghatározott címen kezdődik. A 68020-as 1...32 bit közötti bitmezőket tud feldolgozni. A programozónak természetesen nem kell törődnie a byte-, szó- vagy kettős-szó határokkal, ezt elintézi a 68020-as. A bitmezőhöz való hozzáférési címhez hozzátartozik még mindenkor egy offszet (ami a hozzáférési címtől a bitmező első bitjéig terjedő bitek számát jelenti). Ezt az offszetet (eltolást) egy adatregiszter is meghatározhatja. Ily módon egy offszetre a $-2^{31}\ldots+(2^{31}-1)$ tartományban van lehetőség, ami összesen 256 Mbyte területet jelent.

A bitmezők feldolgozásához nyolc utasítást definiáltak. E nyolc utasítás közül négy megfelel a családhoz tartozó többi processzor bitutasításainak. E mellett még négy olyan utasítást definiáltak, amelyek a bitmezők, regiszterek és a tároló közötti mozgatására alkalmasak. Az adatregiszterekben a mező magasabb helyiértékű bitje (ami a tárban a magasabb helyiértéken áll) a regiszter O. bitjén van. Az alacsonyabb helyiértékű bit (a tárban levő mező első bitje) az adatregiszternek abban a bitjében van, ami a mező eggyel kisebb szélességének felel meg.

A 68020-asnál bármely adattípus bármilyen, tetszőleges byte-címről betölthető. Ez a családhoz tartozó mikroprocesszorokhoz képest igen jelentős bővítést jelent. Adott esetben a 68020-as egymás után több buszhozzáférést is el tud végezni.

Az utasítás-hozzáféréseket a szócímzéseket követően kell elvégezni.

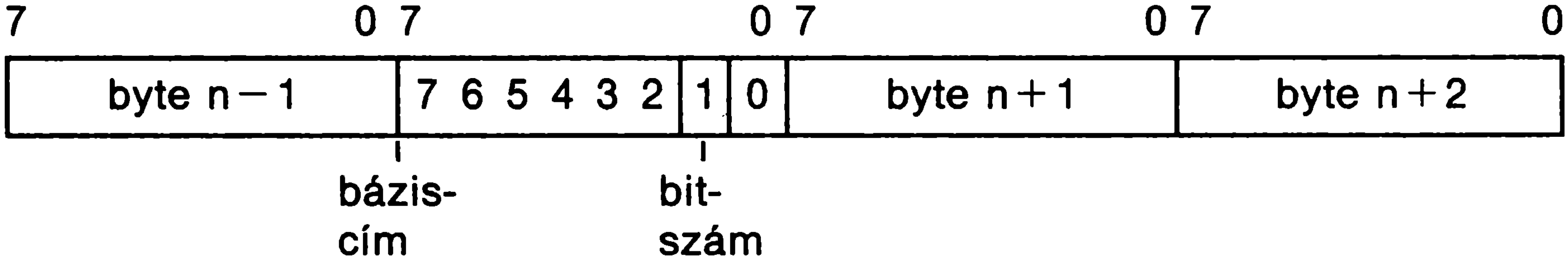
A tárban levő adatok címzése

31 23 15 7 0

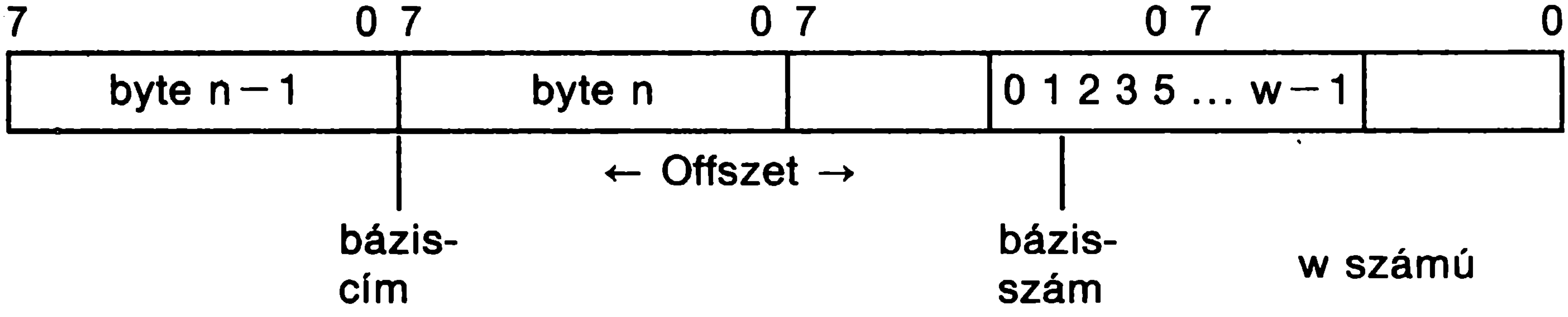
<table>
<tr><td colspan="4">$0000 0000 kettős-szó</td></tr>
<tr><td colspan="2">$0000 0000 szó</td><td colspan="2">$0000 0002 szó</td></tr>
<tr><td>$0000 0000 byte</td><td>$0000 0001 byte</td><td>$0000 0002 byte</td><td>$0000 0003 byte</td></tr>
<tr><td colspan="4">$0000 0004 kettős-szó</td></tr>
<tr><td colspan="2">$0000 0004 szó</td><td colspan="2">$0000 0006</td></tr>
<tr><td>$0000 0004 byte</td><td>$0000 0005 byte</td><td>$0000 0006 byte</td><td>$0000 0007 byte</td></tr>
<tr><td colspan="4">•
•
•</td></tr>
<tr><td colspan="4">$FFFF FFFC kettős-szó</td></tr>
<tr><td colspan="2">$FFFF FFFC szó</td><td colspan="2">$FFFF FFFE szó</td></tr>
<tr><td>$FFFF FFFC byte</td><td>$FFFF FFFD byte</td><td>$FFFF FFFE byte</td><td>$FFFF FFFE byte</td></tr>
</table>

A tárban levő adatok szervezése

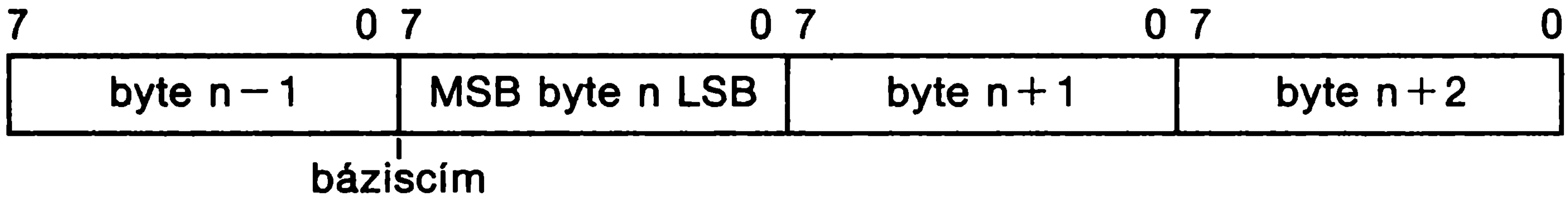
bitadatok


7 0 7 0 7 0 7 0
byte n−1
7 6 5 4 3 2
1
0
byte n+1
byte n+2
bázis-
cím
bit-
szám

bitmezőadatok

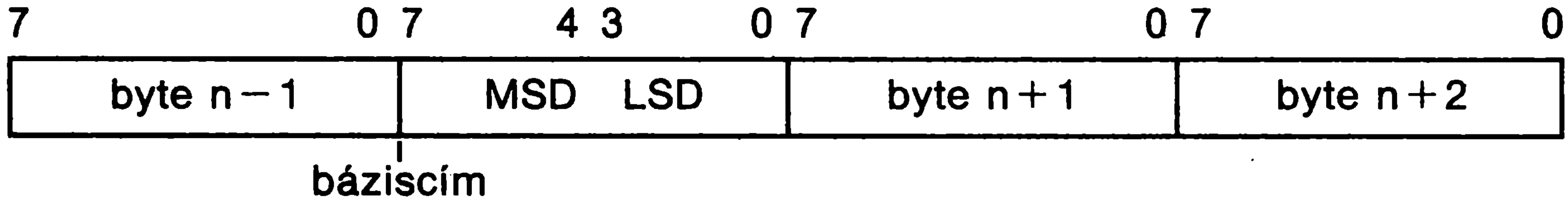

7 0 7 0 7 0 7 0
byte n−1
byte n
0 1 2 3 5 ... w−1
← Offszet →
bázis-
cím
bázis-
szám
w számú

Egész számú adatok (byte)

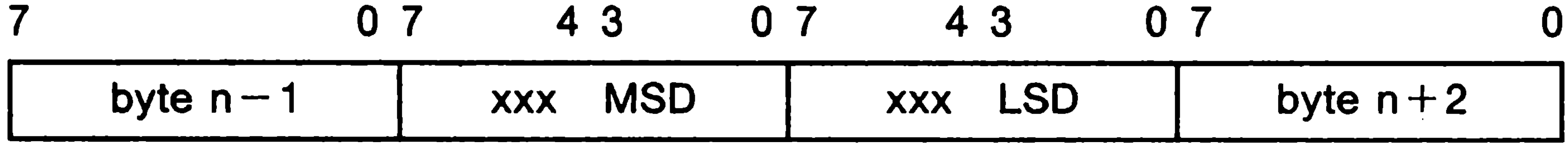

7 0 7 0 7 0 7 0
byte n−1
MSB byte n LSB
byte n+1
byte n+2
báziscím

Szó, kettős-szó, négyes-szó típusú egész számú adatoknál az MSB (Most Significant Bit = nagyobb helyiértékű bit) ugyanazon a helyen van, mint a byte típusú adatoknál. Az LSB (Least Significant Bit = alacsonyabb helyiértékű bit) ennek megfelelően jobbra „vándorol".

BCD adatok, tömörítve


7 0 7 4 3 0 7 0 7 0
byte n−1
MSD LSD
byte n+1
byte n+2
báziscím

BCD adatok, nem tömörítve


7 0 7 4 3 0 7 4 3 0 7 0
byte n−1
xxx MSD
xxx LSD
byte n+2

Pipeline- (csővezeték-) szerkezet és cache-tároló

A 68020-asnak háromlépcsős utasításregisztere van. Az egyes lépcsőket B, C és D betűkkel jelöljük. E szerkezetnek köszönhetően már egy utasítás feldolgozása során betölthetők a következő utasítások vagy kiegészítő szavak. A processzor természetesen szavakat is tölt a belső cache-ből az utasításregiszterbe.

A 68020-as a mindenkori szót először a külső tárból vagy a cache-ből a B lépcsőbe tölti be. Ez aztán a B lépcsőből a C lépcsőn keresztül a D lépcsőbe kerül betöltésre. A D lépcsőben már egy teljesen dekódolt és felülvizsgált szó áll a vezérlőegység rendelkezésére. Azok az utasítások, amelyek kiegészítő szavakat vagy közvetlenül adatokat használnak, ezeket közvetlenül, minden további buszciklus nélkül, a C lépcsőben találják meg. A CPU végrehajtó egysége a C lépcsőhöz is közvetlenül hozzáférhet.

Ha a programokat alaposan elemezzük, akkor kiderül, hogy ezek az idő legnagyobb részét viszonylag rövid ciklusok futására fordítják. Ha meg lehetne valósítani egy olyan eljárást, ami lehetővé tenné azt, hogy ezeket a ciklusokat a processzorban időlegesen tárolni lehessen, akkor ezzel a CPU feldolgozási sebességét lényegesen növelni lehetne.

A 68000-es család a prefetch-mechanizmusokkal a buszt megközelítőleg optimálisan használja ki. Ez különösen a processzor új variánsaira vonatkozik. A busz csak igen ritkán áll a DMA egység (DMA = Direct-Memory-Acces = közvetlen memória-hozzáférés) rendelkezésére. A DMA egységnek a processzor működését valójában fel kell függeszteni ahhoz, hogy a buszt használni tudják. Ez természetesen csökkenti a processzor feldolgozási sebességét, így a DMA-hozzáférések fő előnye gyakran veszendőbe megy.

E két probléma megoldása éppen a 68020-asnál igen fontos, mert a 68020-ast elsősorban a többprocesszoros rendszerekben és társprocesszorokkal együttműködésben felmerülő feladatok megoldására fejlesztették ki. Azért, hogy a processzor ezeket a műszaki előnyöket a többi processzorhoz képest teljes mértékben ki tudja használni, a 68020-as központi chipjén van egy 64 bit szélességű cache-tároló.

A cache-tároló mindegyik mezeje egy kettős-szóból – ami a tároló tulajdonképpeni tartalmát hordozza – és a hozzáférési cím legfelső 24 bitjéből áll. Az A2...A7 címbitek minden egyes lehetséges kombinációjához a cache-tároló pontosan egy tárolója van hozzárendelve (az A0 és A1 címbitek csak egy kettős-szón belül választanak ki egy byte-ot vagy egy szót). Ehhez a cache-tároló minden egyes mezején tartozik egy bit, amelyet a hozzáférési cím 2-es funkciókód bitje (FC2) tartalmaz. Azt, hogy a mező tartalma érvényes-e, egy további bit mutatja meg.

Ha egy utasítás-hozzáférést kell végrehajtani, akkor a 68020-as a keresést először a cache-tárolóban kezdi. Azt, hogy ennek során melyik mezőhöz fordul, az A2...A7 címbitek határozzák meg. Ezután a processzor megvizsgálja, hogy a hozzáférési cím felső 24 bitje és az FC2 bit megegyezik-e a cache-tároló mezejében levő bitekkel. Ha ezen kívül az a bit, amelyik azt mutatja, hogy a mezőben levő adatok érvényesek, aktív, akkor a 68020-as a cache-tárolóból betölti az ehhez szükséges szót. Azt, hogy a cache-tároló mezejéről a két szó melyikét kell betölteni, az A1 vezeték határozza meg.

Ha ennek során a 68020-as a szükséges szót nem a cache-tárolóban találja, akkor ezt egy buszciklussal betölti, és közben aktualizálja a cache-tárolót is. A 68020-as egy utasításszó betöltéséhez szükséges buszciklust alapvetően egy néggyel osztható címre szóló kettős-szó hozzáférésként hajt végre. Ez a cache-tároló aktualizálását leegyszerűsíti.

Azért, hogy a nem érvényes adatok, vagy az adatokkal kapcsolatos egyéb problémák elkerülhetők legyenek, a 68020-as cache-tárolót csak utasítás-hozzáférésekre használja.

A 68020-asnak két regisztere van arra a célra, hogy a felhasználó számára lehetővé tegye a cache-regiszterek vezérlését. Ezeket a regisztereket a MOVEC utasítással lehet megváltoztatni, ill. olvasni.

A cache-címregiszter (CAAR) a cache-tárolónak azt a mezejét határozza meg, amelyet mindenkor egy Clear entry utasítás töröl (egy egyedi mezőre vonatkozó törlőutasítás, amelyet a cache-control regiszter vált ki). Itt a cache címregiszter 2...7 bitje határozza meg azt, hogy melyik mezőt kell törölni. A 8...31 biteknek nincs szerepük.

A cache-control regiszter felépítése:

Bit	**Funkció**
0	Enable Cache (engedélyez) Ha a bit értéke 1, akkor a cache-tároló be van kapcsolva. Egy külső reset-jelet követően ez a bit többnyire törlődik.
1	Freeze Cache (befagyaszt) Ha a bit értéke 1, akkor a cache-tárolóból csak olvasni lehet, beleírni nem. Ezt a bitet az emulátorszoftverek használhatják arra, hogy a felhasználói program számára megkapják a cache-tároló tartalmát.
2	Clear Entry (belépés törlés) Ha a bitet magasra állítjuk, akkor a cache-tárolónak az a mezeje, amelyet a cache címregiszter 2...7 bitje határoz meg, törlődik.
3	Clear Cache (cache törlése) E bit magasra állításával a cache-tároló teljes tartalmát töröljük. (Illetve pontosabban fogalmazva érvénytelennek nyilvánítjuk.)
4...31	Nem használt. Ezek a bitek, valamint a 2. és a 3. bit olvasáskor mindig nulla értékűek.

Ezen kívül a CPU-nak van még egy bemenete, ami a cache-tároló kikapcsolására alkalmas. Ennek a bemenetnek a jele nCDIS. A cache-tároló – a cache-control regiszter (CACR) O. bitjétől függetlenül – az nCDIS aktiválását (alacsonyra állítását), és a belső szinkronizálást követő első cache-hozzáféréstől kezdve (mint valamennyi aszinkron bemenetnél) ki van kapcsolva. Ezt a bemenetet külső emulátorhardvernek használhatjuk. Egy emulátor csak akkor tudja az összes buszhozzáférést figyelemmel kísérni, ha a cache-tároló ki van kapcsolva.

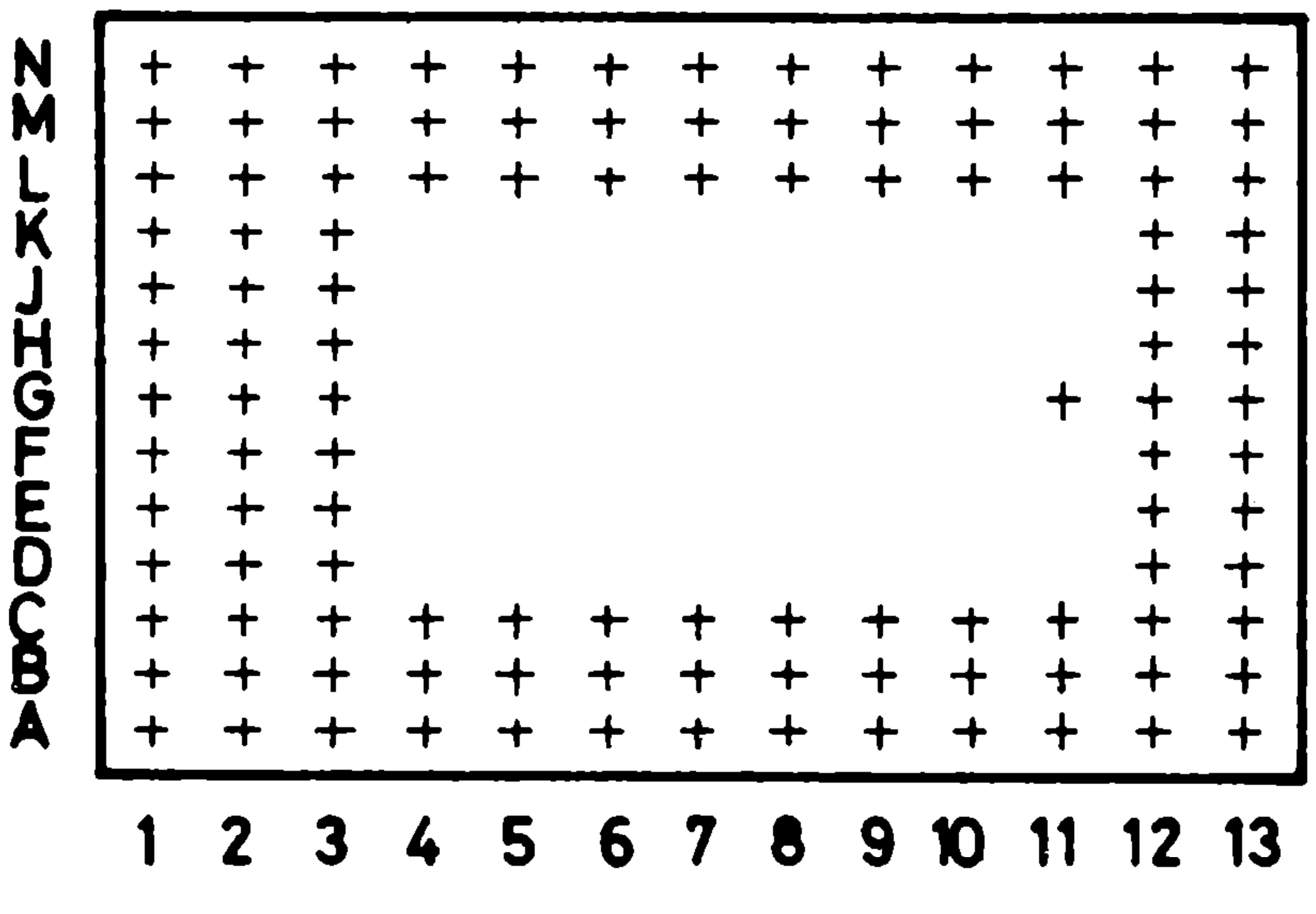
Egy processzor-reset során mindkét cache-regiszter (a CACR és a CAAR) nullára állítódik. A tulajdonképpeni cache-tároló tartalma érvénytelennek minősül.

A 68020-as jeleinek és buszvonalainak ismertetése

A 68020-ast egy 114 csatlakozójú, PIN–GRID tokozásban forgalmazzák. Ez a tokozási forma lehetővé teszi, hogy valamennyi csatlakozót bármely vezeték multiplex módon való használata nélkül be lehessen kötni.

A következőkben bemutatjuk a 68020-as lábkiosztását. A tokozás alakja a valóságban négyzetes (mint az egyéb típusú PIN–GRID tokozásnál). A lábak egymás közötti távolsága 2,54 mm (0,1 coll).

A csatlakozólábak ismertetésénél az alacsony-aktív (LOW) jeleket vagy a jelek alacsony-aktív (LOW) állapotát ismét egy n betűvel jelöltük. Azokat a megnevezéseket, amelyekre magyar [az eredeti szöveg szerint „német” – Ford. megjegyzése] nyelvű megnevezés nemigen található, meghagytuk eredetiben.



A 68020-as lábkiosztása (PIN–GRID tokozás)

Csatlakozóláb száma	Megnevezése
A1	nBGACK
A1	nBGACK
A1	nBGACK
A1	nBGACK
A2	A1
A3	A31
A4	A28
A5	A26
A6	A23
A7	A22
A8	A19
A9	V_{cc}
A10	GND
A11	A14
A12	A11
A13	A8
B1	n.c.
B2	nBG
B3	nBR
B4	A30
B5	A27
B6	A24
B7	A20
B8	A18
B9	GND
B10	A15
B11	A13
B12	A10
B13	A6
C1	nRESET
C2	CLK
C3	n.c.
C4	A0
C5	A29
C6	A25
C7	A21
C8	A17
C9	A16
C10	A12
C11	A9
C12	A7
C13	A5

Csatlakozóláb száma	Megnevezése
D1	V_{cc}
D2	V_{cc}
D3	n.c.
D12	A4
D13	A3
E1	FCO
E2	nRMC
E3	V_{cc}
E12	A2
E13	nOCS
F1	SIZ0
F2	FC2
F3	FC1
F12	n.c.
F13	nIPEND
G1	nECS
G2	SIZ1
G3	nDBEN
G11	V_{cc}
G12	GND
G13	V_{cc}
H1	nCDIS
H2	nAVEC
H3	nDSACK0
H12	nIPL2
H13	GND
J1	nDSACK1
J2	nBERR
J3	GND
J12	nIPL0
J13	nIPL1
K1	GND
K2	nHALT
K3	n.c.
K12	D1
K13	D0
L1	nAS
L2	R/nW
L3	D30
L4	D27
L5	D23
L6	D19

Csatlakozóláb száma	Megnevezése
L7	GND
L8	D15
L9	D11
L10	D7
L11	n.c.
L12	D3
L13	D2
M1	nDS
M2	D29
M3	D26
M4	D24
M5	D21
M6	D18
M7	D16
M8	V_{cc}
M9	D13
M10	D10
M11	D6
M12	D5
M13	D4
N1	D31
N2	D28
N3	D25
N4	D22
N5	D20
N6	D17
N7	GND
N8	V_{CC}
N9	D14
N10	D12
N11	D9
N12	D8
N13	n.c.

Címbusz (A0, A1, A2...A31)

A címbusz normál buszciklusok során a hozzáférési címet tartalmazza. Az A0 és az A1 itt különleges szerepet játszik. Ha azon a címen, amire a hozzáférés irányult, kettős-szó tárolóhely kezdődik, akkor az A0 és az A1 a megcímzett társzón belül egy byte-offszetet jelöl. Ha viszont a tár (vagy a periféria) megszólított címe szó szélességű, akkor az A1 egy normál címet és az A0 egy offszetet jelent. Csak byte-szervezésű tár (vagy periféria esetén szolgáltatja az A0...A31 a teljes címet. Egy CPU-space-ciklus alatt a címbusz más információkat továbbit. A címbusz meghajtói háromállapotú (tri-state) meghajtók.

Adatbusz (D0...D31)

Az adatbusz négy, egyenként 8 bit szélességű részre van felosztva. Az adatbusz meghajtói kétirányú, háromállapotú (bidirectional, tri-state) meghajtók.

Átvitel hosszúsága (SIZ0, SIZ1)

Ezzel a két kimenettel azt jelzi a CPU, hogy ezzel a ciklussal hány byte-ot kíván olvasni, ill. írni:

SIZ1	SIZ0	Az átvitel hosszúsága
0	1	byte
1	0	szó
1	1	3 byte
0	0	kettős-szó

External Cycle Star
(nECS = külső ciklusindítás)
Ezt a kimenetet a 68020-as akkor aktiválja, ha elkezdődik egy buszciklus, de ezt megelőzően nem világos, hogy a keresett szó a cache-tárolóban van-e. Ezért ezt a jelet a buszciklus során az nAS jellel össze kell hasonlítani.

Operand Cycle Start (nOCS)
Ennek a kimenetnek ugyanaz a funkciója, mint az nECS-nek. Viszont csak akkor kerül aktiválásra, ha a processzor az elindított buszciklusban egy operandust ír vagy olvas.

Read-Modify-Write ciklus (nRMC)
A 68000-esnél egy olvasás-módosítás-írás ciklus során a két hozzáférés között az nAS nem lett kikapcsolva. A 68020-asnál a két hozzáférés között az nAS kikapcsolásra kerül. Azért viszont, hogy az olvasás-módosítás-írás ciklust mégis tudja jelezni, a 68020-as ezt a kimenetet aktiválja.

Cím engedélyezése (nAS)
Ha ez a kimenet aktív, akkor egy érvényes cím van a címbuszon.

Adat engedélyezése (nDS)
A CPU egy olvasási ciklusa során ez a kimenet azt jelzi, hogy a külső egység mikor adja be az adatokat. Írási ciklus során a CPU akkor aktiválja ezt a kimenetet, ha a buszon érvényes adatok vannak.

Olvasás/írás (R/nW)
Ennek a vezetéknek ugyanaz a funkciója, mint az összes többi processzornál.

Adatpuffer engedélyezése (nBEN)
Ez a háromállapotú kimenet egy külső adatpuffert kapcsol be, erre a kimenetre nincs minden rendszernél szükség.

Adatok átvitelének és hosszúságának nyugtázása
(nDSACK0, nDSACK1)
Ennek a két bemenetnek ugyanaz a funkciója, mint az nDTACK bemenetnek a család többi processzorainál. Ezen túlmenően ezek azt is jelzik, hogy a tárolók, ill. a perifériák a hozzáférési címen hány bit szélességűek, (ez az ún. kapuszélesség) tehát, hogy valójában hány bit átvitelére került sor.

nDSACK1	nDSACK0	Eredménye
1	1	várakozó ciklusokat beiktatni
1	0	az adatbusz 8 bit szélességű *)
0	1	az adatbusz 16 bit szélességű *)
0	0	az adatbusz 32 bit szélességű *)

* buszciklust befejezni

Ha az adatbusz 8 bit szélességű, akkor az adatokat a D31...D24 vezetékekre kell helyezni, ill. onnan átvenni (D7-est a D31-esre). Egy 16 bit szélességű adatbusz esetén a D31...D16 vezetékek kerülnek felhasználásra (D16-ot a D31-re kapcsolni, ill. onnan átvenni. Csak a 32 bit szélességű adatbusz esetén kerül a teljes busz alkalmazásra.

A 68020-as az nDSACK0 és az nDSACK1 bemeneteket belsőleg szinkronizálja. A jelek időben egy fél rendszerütemidővel egymáshoz képest el lehetnek tolódva.

Cache kikapcsolása (nCDIS)
Ezzel a bemenettel lehet a belső cache-tárolót kikapcsolni.

Megszakítás-prioritás bemenetek (nIPLO...2)
Ezeknek a bemeneteknek a szerepe ugyanaz, mind a család többi processzorainál.

Megszakítás elfogadása (nIPEND)
Ezt a kimenetet akkor aktiválja a 68020-as mihelyt elfogad egy megszakítást, tehát a még futó busz- vagy utasításciklus közben. Ez a kimenet a tulajdonképpeni megszakításnyugtázás előtt aktiválásra kerül.

Autovektor-megszakítás (nAVEC)
Mivel a 68020-asba szinkron buszhozzáférést nem építettek be, így nVPA csatlakozó sincs, amivel egy autovektor-megszakítást lehetne jelezni. Az nVPA bemenetnek ezt a funkcióját a 68020-asnál az nAVEC bemenet veszi át.

Buszvonal átadása (nBR, nBG, nBGACK)
A buszvonal átadása a 68020-asnál elvileg ugyanúgy történik, mint a 68000-esnél.

Rendszervezérlés (nRESET, nHALT, nBERR)
Ezeknek a bemeneteknek a 68020-asnál ugyanazok a funkciói, mint a család többi processzorainál. A CPU-nak azonban a többértelműségekre és extrém időpontokra való reagálásai a bemenetek aktiválásakor kismértékben módosultak.

Rendszerütem és tápfeszültség

A 68020-as rendszerüteme megegyezik a többi processzoréval. A 68020-as azonban a 8 és 12,5 MHz, ill. 8 és 16,7 MHz közötti rendszerütem-frekvenciákra alakították ki.

A feszültségellátás +5 V és a földcsatlakozón keresztül történik, 3 csoportra osztva. Így egymástól független a tápfeszültség-ellátása a cimbuszpuffernek, az adatbuszpuffernek, valamint a többi kimenetnek és a belső logikai köröknek. A három feszültség vonatkoztatási pontjának azonban közösnek kell lennie.

A 68020-as buszműveletei

A 68020-asnál a buszhozzáférést meggyorsították. Míg a 68000-esnél egy buszhozzáférésnek minimum 4 rendszerütemciklus kellett, a 68020-asnál ehhez már három is elegendő. A 68020-asnál a buszcikluson belül a kimenetek korábban aktiválódnak, mint a 68000-esnél. Ezért az nDSACK0 és az nDSACK1 vezetéket már az S2 lefutó élétől kezdődően le lehet kérdezni, és egy ciklust már az S5 után be lehet fejezni. A 68000-estől eltérően itt már az nAS az első jel, ami egy buszhozzáférés lefolyásakor aktiválásra kerül. Ez csak a rendszerütem S1 félciklusában jelenik meg. Lényegesen korábban (az S0-ban) aktiválódik már az nECS, ill. az nOCS jel, és szinte valamennyi többi jel is.

Ugyanúgy, mint a 68000-esnél, a 68020-asnál is várakozó ciklusok vannak beiktatva mindaddig, míg a buszhozzáférés nyugtázásra nem kerül. Az nDSACK0 és az nDSACK1 jelekkel való nyugtázás egyben a csatlakoztatott tár (ill. periféria) kapuszélességét (pot) is a processzor tudomására hozza.

A 68020-as az offszettől és az átviteli szélességtől (byte, szó, 3 byte vagy kettős-szó) függően helyezi a négy adatbuszterületre (D31...D24, D23...D16, D15...D8 és D7...D0) az átviendő adatokat.

Az OP0 mindig az operandus 31...24. bitjeit, az OP1 a 23...16. bitjeit, az OP2 a 15...8. bitjeit, végül az OP3 az operandus 7...0. bitjeit jelentik. A 68020-asban egy belső multiplexer gondoskodik arról, hogy az operandus mindenkori megfelelő része az adatbusz megfelelő helyére kerüljön.

A 68020-as egy operandus „maradék”-át, azaz az operandus azon részeit, amit az első buszhozzáférés során nem tudott átvinni, automatikusan további buszciklusokban viszi át. Az ilyen jellegű maradékok akkor jönnek létre, ha szavakat vagy kettős-szavakat páratlan címekre kell átvinni, vagy ha a tár kapuszélessége nem elegendő az operandushoz.

A belső adatmultiplexer működése:

Átviteli			**Cím**		**Az adatbuszvezetékek tartalma**			
szélesség	**SIZ1**	**SIZ0**	**A1**	**A0**	**31...24**	**23...16**	**15...7**	**7...0**
byte	0	1	x	x	OP3	OP3	OP3	OP3
szó	1	0	x	0	OP2	OP3	OP2	OP3
	1	0	x	1	OP2	OP2	OP3	OP2
3 byte	1	1	0	0	OP1	OP2	OP3	OP1
	1	1	0	1	OP1	OP1	OP2	OP3
	1	1	1	0	OP1	OP2	OP1	OP2
	1	1	1	1	OP1	OP1	OP1	OP1
kettős-szó	0	0	0	0	OP0	OP1	OP2	OP3
	0	0	0	1	OP0	OP0	OP1	OP2
	0	0	1	0	OP0	OP1	OP0	OP1
	0	0	1	1	OP0	OP0	OP1	OP0

x: tetszőleges

A 68020-asnak a rugalmas buszkoncepciója, az aszinkron, dinamikus buszhozzáférés következtében nagy előnyei vannak. Már maga az a tény, hogy a tárban az adathozzáféréseknél nem kell a szó-, ill. kettős-szó határokra ügyelni, jelentősen leegyszerűsíti a 68020-as programozását. Ezeket az előnyöket csak azon az áron lehetett elérni, hogy a dekódoláshoz megnőtt a külső hardverigény.

A dekódoláshoz számos kapuegységre van szükség. Ezeknek a logikai funkciója azonban egy PAL-egységgel, azaz egy programozható logikai egységgel is megvalósítható. A logikai kapcsolás egyszerűbb akkor, ha a rendszerben nem fordul elő valamennyi lehetséges kapuszélesség.

Aktív adatbusztartományok az egyes kapuszélességek esetén:

A belső adatmultiplexer működése:

Átviteli			**Cím**		**Az adatbusz aktív tartományai**			
szélesség	**SIZ1**	**SIZ0**	**A1**	**A0**	**31...24**	**23...16**	**15...7**	**7...0**
byte	0	1	0	0	B, W, L	–	–	–
	0	1	0	1	B	W, L	–	–
	0	1	1	0	B, W	–	L	–
	0	1	1	1	B	W	–	L
szó	1	0	0	0	B, W, L	W, L	–	–
	1	0	0	1	B	W, L	L	–
	1	0	1	0	B, W	W	L	L
	1	0	1	1	B	W	–	L

Átviteli	**SIZ1**	**SIZ0**	**Cím**		**Az adatbusz aktív tartományai**			
szélesség			**A1**	**A0**	**31...24**	**23...16**	**15...7**	**7...0**
3 byte	1	1	0	0	B, W, L	W, L	L	–
	1	1	0	1	B	W, L	L	L
	1	1	1	0	B, W	W	L	L
	1	1	1	1	B	W	–	L
kettős-szó	0	0	0	0	B, W, L	W, L	L	L
	0	0	0	1	B	W, L	L	L
	0	0	1	0	B, W	W	L	L
	0	0	1	1	B	W	–	L

B: a kapu szélessége byte (8 bit)
W: a kapu szélessége szó (16 bit)
L: a kapu szélessége kettős-szó (32 bit)

Példa

Egy kettős-szó hozzáférést kell egy szó szélességű tárolóhelyre elvégezni (A0 = 1, A1 = 0). Az első ciklusban egy kettős-szó hosszúságú ciklus kerül végrehajtásra (A0 = 1, A1 = 0). Ezt nyugtázza a tároló, amiről a CPU felismeri, hogy egy byte-átvitelre került. Ezt a CPU-nak egy három byte-os ciklusa követi, ennek során az A0 és A1 nulla. A teljes cím (A0...A31) 1-gyel megnő. A tároló nyugtázásából a 68020-as észleli, hogy egy szóátvitelre került. Az utolsó byte-ot a 68020-as egy harmadik ciklusban viszi át. Előtte azonban a hozzáférési címet kettővel megnöveli, az A0 és A1 tehát ismét nulla.

CPU-space-ciklusok

A 68020-as funkciókód-vezetékeinek némileg megváltozott a jelentése:

FC2	FC1	FC0	Jelentés
0	0	0	nem használt, foglalt
0	0	1	user- (felhasználói) adatok
0	1	0	user- (felhasználói) program
0	1	1	nem használt, a felhasználó számára foglalt
1	0	0	nem használt, foglalt
1	0	1	supervisor-adatok
1	1	0	supervisor-program
1	1	1	CPU-space

A 68020-asnál négy különböző CPU-space-ciklus van impementálva. A funkciókód-vezetékek %111 kombinációja a 68000-esnél egyértelműen egy megszakítási nyugtázást jelent. Ahhoz, hogy az egyes CPU-space-ciklusokat egymástól meg tudjuk különböztetni, a 68020-asnál ezen túlmenően még az A19...A16 vezetéket is dekódolni kell.

A CPU-space-ciklusok társprocesszorokkal való kommunikálásra, a tárkezelő egységek (MMU) vezérlésére, a töréspontok (breakpoints) kijelzésére és a megszakítások nyugtázására alkalmasak.

A következő táblázat a címbusznak a különböző CPU-space-ciklusok során való használatát mutatja.

A megszakítások nyugtázása a 68020-asnál elvileg ugyanúgy megy végbe, mint a 68000-esnél. Mivel azonban a 68020-asnál szinkron buszhozzáférés, és ennélfogva nVPA vezeték nincs, beépítették az nAVEC bemenetet. Ha egy megszakítás nyugtázása során ez a bemenet aktiválódik, akkor a processzor egy autovektor megszakítást hajt végre.

Egy nem autovektor megszakításnál a felhasználónak először a kívánt vektorszámot az adatbuszra kell helyeznie, és ezt követően aktiválhatja az nDSACK0, ill. nDSACK1 vezetékeket. Ezekkel a vezetékekkel jelzi a CPU ebben az esetben is az adatbusz kapuszélességét. Az nDSACK0 és az nDSACK1 kombinációjától függően veszi elő a 68020-as a vektort az adatbusz különböző részeiből. Ennek során a választott adatbuszszélesség (kapuszélesség) legalsó (alacsonyabb helyiértékű) 8 bitjét olvassa be.

A 68010-eshez hasonlóan a 68020-asnál is van lehetőség arra, hogy töréspontokat (breakpoints) helyezzünk el a programban. A 68020-as azonban a töréspontcikluson belül a töréspont számát is jelzi. A töréspont számától függően a felhasználó a töréspont kétféle feldolgozási módja között választhat:

1) A felhasználó aktiválja az nBERR-t. Erre a processzor végrehajt egy normál kizárási feldolgozást. Ehhez betölti az illegális utasítás (illegal instruction) vektort.

2) A felhasználó aktiválja az nDSACK0-át, ill. az nDSACK1-et. Ebben az esetben a processzor beolvassa a D31...D16 adatvezetéket (ha a kapu szélessége 16 bit) vagy kétszer a D31...D24 vezetéket (ha a kapu szélesség 8 bit). A beolvasott adatokat kicseréli a töréspont-utasítással, és megkísérli azokat utasításként értelmezni. Így a program végrehajtása közvetlenül folytatódhat.

Az A19...A16 címvezetékeken levő %0001 kombinációjú CPU-space-ciklusok általában egy, a rendszerben levő MMU-hoz szólnak. Az ilyen jellegű hozzáférések a CALLM (CALL Module) és az RTM (ReTurn from Modul) utasítások feldolgozása során történnek. A 68020-as modulkoncepciójának keretében tervbe van véve, hogy egy MMU-t közvetlenül lehessen vezérelni.

	funkciókód			címbusz																															
	2	1	0	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
breakpoint (töréspont) visszaigazolása	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	BRKPT Nr			0	0
tárkezelő egység (MMU)	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	MMU-Register						
kommunikáció a társprocesszorokkal	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	CPID			0	0	0	0	0	0	0	0	CP-Register				
megszakítás-visszaigazolás	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	Int-Prior			1

BRKPT Nr.: breakpoint száma
CPID : társprocesszor azonosító száma
Int-Prior : megszakítás prioritási fokozata

CPU-Space hozzáférési mód

Az A19...A16 címbitek összes többi kombinációját a 68020-as nem használja, ezek az új típusokhoz, későbbi alkalmazásra vannak lefoglalva.

CPU-space címdekódolás

Az A19...A16 címvezeték negyedik alkalmazott kombinációjával egy CPU-space-ciklus során adatokat és utasításokat lehet a 68020-as és a társprocesszorok között átvinni.

Kizárások feldolgozása a 68020-asnál

A 68020-asnál valamennyi, a 68000-esnél meglevő kizárás megtalálható. Ezért a kizárási táblázat ugyanaz, mint amit a többi processzor is használ. A 68010-es formátumhiba kizárása is létezik. Ez azonban a modulfeldolgozásnál és a társprocesszor-utasításoknál bizonyos hibahelyzetekben is kiváltásra kerülhet.

A 68020-asnál a kizárások feldolgozásához több mint három prioritási fokozatot definiáltak. Valamennyi kizárás öt fő csoportra osztott.

Számos kizárás a 68000-eshez és a 68010-eshez képest több adatot helyez el a verembe. A 68020-as a 68010-eshez hasonlóan egy formátumkódot helyez a verembe. A 68020-asnál a formátumbitek összesen hét érvényes kombinációja létezik, amelyek a hat lehetséges veremformátum közül mindig kiválasztanak egyet. RTE utasításnál a 68020-as is felülvizsgálja a formátumot. Ha ennek során hibát állapít meg, akkor a 68020-as a formátumhiba-kizárást váltja ki.

Egy új kizárást építettek be a 68020-asba. Ha társprocesszorral való kommunikáció során protokollhiba lép fel, akkor a protokollhiba-kizárás kerül kiváltásra. Ennek a vektorszáma $0D, a vektor a vektor-bázis-regiszter plusz $034 címen található.

A trace kizárás megváltozott. A 68020-asnál az állapotregiszterben a trace funkciók bekapcsolására két bit áll rendelkezésre. Vagy nem lép fel trace kizárás, vagy minden egyes utasítás után fellép a trace kizárás, vagy csak akkor kerül sor kizárásra, ha a programszámláló a program futása során ugrásszerűen megváltozik, ha tehát egy ugrás előre vagy vissza fordult elő. A T1 és T0 bitek felhasználása a következő:

T1	**T0**	**Jelentése**
0	0	nincs trace kizárás
0	1	trace ugrásnál, visszaugrásnál stb.
1	0	trace minden egyes utasítás után
1	1	nem használt, fenntartott

Ezzel a bővítéssel a 68020-as trace funkciója lényegesen nagyobb teljesítményű. Különösen a nagy programokban fáradságos a lépésenkénti munka. Többnyire csak azok a helyek érdekesek, amelyeken a program menete megváltozik. Ezek általában azok a helyek, ahol alprogramokat hívunk be vagy fejezünk be.

A 68020-asnál a CHK (CHecK = megvizsgálni) utasítás mellett – amellyel regisztertartalmakat lehet vizsgálni – létezik még a CHK2 utasítás. Ezzel regisztertartalmakat lehet vizsgálni egy felső és egy alsó határig. Negatív vizsgálati eredmény esetén mindkét utasítás a CHK-kizárást váltja ki (száma: $06, vektor: $018).

A 68000-es TRAPV utasítása mellett – amellyel az állapotregiszterben levő túlcsordulásbit (V) állapotától függően lehetett kizárást kiváltani – a 68020-asnál létezik még a cpTRAPcc, TRAPcc utasítás, amelyek ugyanazt a kizárást váltják ki (száma: $07, vektor: $01C). A TRAPcc utasítással az állapotregiszterben levő kapcsolók tetszőleges feltételétől függően (mint pl. az elágazási utasításoknál) lehet a kizárást kiváltani. A cpTRAPcc az ennek megfelelő utasítás a társprocesszorok esetén. Itt is megadható egy feltétel (cc, pl. "EQ" = EQual, egyenlőség). Ilyenkor azonban nem a 68020-as állapotregiszterében levő kapcsolókat vizsgálja, hanem ehelyett a társprocesszort kérdezi le az adott feltétel teljesülését illetően.

Ugyanúgy, mint a 68010-esnél, a 68020-asnál is privilegizált a teljes állapotregiszter tartalma. Ezáltal virtuális processzorkoncepciók valósíthatók meg. A 68020-asnál is be van építve a felhasználói részbe a MOVE from CCR utasítás.

A 68020-asnál lehetőség van arra, hogy azokat az utasításokat, amelyek töréspontokat váltanak ki, egy CPU-space-ciklus közben a cím birtokában megkülönböztessük. Itt a $4848 (0. töréspont) és a $484F (8. töréspont) közötti utasítások váltják ki az egyes töréspontokat.

A 68000-esnek minden utasítása, amelynek utasításszava %1111-gyel ($F) kezdődött – tehát $Fxxx alakú – különleges kizáráshoz vezet. A $Fxxx alakú utasítások a 68020-asnál társprocesszor utasításokként szolgálnak. Ezért nem vezetnek feltétlenül a „$Fxxx alak nem implementált utasítása" nevű kizáráshoz (kizárás száma $0B). Ha egy CPU-space-ciklusra egy társprocesszor sem válaszol, és az nBERR jel aktív, akkor a 68020-as feltételezi, hogy egy társprocesszor sem üzemel, és kiváltja a $OB kizárási számú, „$Fxxx alak nem implementált utasítás" nevű kizárást (vektor a vektor-bázis-regiszter plusz $02C-nél). A vonatkozó kizárási rutinban pl. a társprocesszor funkciója emulálható.

A 68020-asnál is van lehetőség arra, hogy buszhiba vagy címhiba esetén a hibás ciklus vagy megismételjük, vagy szoftver útján emuláljuk. Ehhez információkat kell a verembe helyezni. Az erre vonatkozó eljárás lényegében megegyezik a 68010-esnél alkalmazott eljárással. Ha a 68020-asnál az utasításfeldolgozás elején lép fel buszhiba, akkor nem kerül a teljes belső állapot tárolásra. Ez csak akkor történik meg, ha a hiba az utasítás feldolgozás közepén lép fel. Ezért a buszhibáknál két különböző veremformátum van.

A 68020-asnál címhiba kizárás csak abban az esetben kerül kiváltásra, ha az utasítást páratlan számú címről kísérelünk meg betölteni. A vonatkozó kizárási rutinban például a nem megengedett hozzáférés emulálható. Egy operandusnál páratlan számú címre való hozzáféréskor már nem kerül kizárás kiváltásra, mert az ilyen jellegű hozzáférések végrehajtódnak.

A 68020-as társprocesszor interfésze (csatlakozó felület)

Az, hogy a 68020-ast társprocesszorokkal együtt lehet működtetni, új típusú adatok vagy műveletek bevezetését teszi lehetővé. Ezzel a rendszer teljesítőképessége lényegesen növelhető.

A társprocesszor interfész lényegében arra szolgál, hogy a fő- és társprocesszorok közötti szinkron műveleteket támogatni lehessen. A koncepció néhány részlete az aszinkron üzemmódot is lehetővé teszi, ez azonban lényegében nem felel meg a 68020-as társprocesszor koncepciójának. A 68020-as társprocesszor koncepciójánál a hangsúly a flexibilitáson volt. Csatlakoztathatók DMA-t (közvetlen tárhozzáférés) kezelő, valamint DMA-t nem kezelő processzorok is. Nem a koncepció határozza meg azt, hogy a társprocesszoroknak milyen jellegű legyen a feladatuk vagy az utasításkészletük.

Társprocesszorokat a család többi tagjaihoz is lehet csatlakoztatni. Mivel azonban ezeknek valójában nincs saját társprocesszor interfészük, az ehhez szükséges műveleteket szoftver útján kell emulálni. Ez azonban nem jelent nagy problémát. Valamennyi társprocesszor utasítás $Fxxx alakú. Ha megkíséreljük, hogy egy ilyen alakú utasítást dolgozzunk fel, akkor kizárás kerül kiváltásra. Ugyanez a kizárás kerül kiváltásra a 68020-asnál, ha nincs társprocesszor, amelyik válaszolna. A megfelelő kizárási rutinban csak a társprocesszor funkcióját kell emulálni. Akár az egész társprocesszor is emulálható. A rendszer továbbfejlesztése során a szoftveroldali emulátorrutinokat a későbbiekben hardverbe épített társprocesszorral lehet helyettesíteni. Ez a magasabb szintű (felhasználói) programokat nem változtatja meg.

A társprocesszor alapvetően hat állapot valamelyikében van. Vagy teljesen szabad vagy szabad az új utasítások fogadására, tartalmaz azonban a legutolsó műveletből eredményeket, ill. egy kizárási felkérést, vagy teljesen foglalt ('busy'). A foglalt állapot tovább bontható. Különbséget teszünk aközött, hogy a társprocesszor azért foglalt, mert csupán további információkra (pl. operandusra) van szüksége, vagy mert ezek befogadására már készen áll.

Minden egyes társprocesszor-utasításban a lehetséges nyolc társprocesszor egyikét kiválasztjuk. Az utasítás mind a négy felső bitjének értéke 1. A 11...9. bitek tartalmazzák a megszólítandó társprocesszor számát, az ún. társprocesszor-azonosítót (Cp-Id). A %000...%101 társprocesszor-azonosító számokat a Motorola lefoglalta saját részére. Ezek közül a %000-át és a %001-et már felhasználták. A %001 azonosító általában az MC68881 lebegőpontos társprocesszorhoz használatos. Az MC68851 jelű MMU a %000 azonosítót már a chipen dekódolja, így ennek a rendszerben mindig a 0. társprocesszornak kell lennie.

A társ- és a főprocesszor közötti átvitelt mindig az utóbbi kezdeményezi. Ennek során a buszátvitelt CPU-space-ciklusokkal végzi. Valamennyi társprocesszor számára egy alapprotokoll kerül felhasználásra a társ- és a főprocesszor közötti kommunikálásra:

1) A kommunikációt a főprocesszor kezdi meg, amennyiben egy CPU-space-ciklussal információkat visz át a társprocesszor regiszterébe.

2) A főprocesszor elolvassa az erre az információra adott választ (Response):

a) A társprocesszor azt jelzi, hogy még foglalt. A főprocesszornak még egyszer meg kell kérdeznie a társprocesszort. Ezáltal lesznek a fő- és a társprocesszor közötti akciók szinkronban.

b) A válasz egy kizárást jelez. A főprocesszor végrehajtja ezt a kizárást.

c) A válasz azt jelzi, hogy a társprocesszornak szüksége van a főprocesszorra pl. azért, hogy adatokat vihessen át. Ennek során a társprocesszor még azt is jelezheti, hogy ezen akció után további kérdést vár a főprocesszortól.

d) A társprocesszor azt jelzi, hogy a munkát befejezte, és már nincs szüksége a főprocesszorra. A főprocesszor nekiláthat a következő utasítás feldolgozásának.

Általánosságban úgy lehet fogalmazni, hogy a főprocesszor átad egy utasítást a társprocesszornak. Válaszként erre az utasításra a főprocesszor elolvassa a társprocesszor válaszát (Response) egy CPU-space-ciklussal. A társprocesszor válaszától függően folytatódik az utasítás végrehajtása. A társprocesszor válaszát a főprocesszor részére adott primitív utasításként is fel lehet fogni. Ezért a válaszadásnak ezt a fajtáját gyakran *primitív*nek is nevezik.

Valamennyi társprocesszor részére rögzítve van egy alap regiszterstruktúra. Erre azért van szükség, hogy lehetővé váljon a processzorok közötti szabályozott kommunikáció. Ez az alap regiszterstruktúra tartalmazza az utasítások, a válasz (Response), az adatok regiszterét, valamint egy vezérlő regisztert és egy kapcsolókat (flageket) tartalmazó regisztert. A kapcsolókat tartalmazó regiszter funkcióját illetően megfelel az állapotregiszter user-byte-jának. A 68020-as azonban a társprocesszor kapcsolóit saját maga nem kérdezi le akkor, amikor azt ellenőrzi, hogy egy feltétel teljesül-e. Csak egy feltételkódot visz át, és igaz vagy hamis választ kap.

A következő társprocesszor-utasítások léteznek:

cpBcc
cpDBcc
cpGEN
cpRESTORE
cpSAVE
cpScc
cpTRAPcc

Ezen utasítások ismertetésére az Utasításkészlet című fejezetben kerül sor.

Ezenkívül van a társprocesszorok 18 primitívje, amelyeket a következő fő csoportokba lehet sorolni:

1) Processzorszinkronizálás
2) Utasításátvitel
3) Kizárás feldolgozása
4) Általános operandusátvitel
5) Regiszterátvitel

A társprocesszorokat egy primitív válasszal le lehet kérdezni arról, hogy éppen supervisor-állapot van-e aktiválva. Ily módon a társprocesszor-műveletek privilégizálhatók.

A kizárásokat a társprocesszorok különböző módon válthatják ki. A társprocesszor utasíthatja a főprocesszort arra, hogy vagy ugorjon a kizárási rutinra, vagy előtte tárolja az állapotát a verembe, hogy később lehetővé váljon az utasítás végrehajtásának folytatása. Ha a kizárási feltétel egy utasítás végrehajtásának az elején lép fel, akkor a társprocesszor arra utasítja a főprocesszort, hogy rövid veremformátumot válasszon. Ha viszont a hiba egy utasítása közepén lép fel, akkor a főprocesszornak a közbenső állapotot kell megőriznie. Még akkor is, ha a hiba a társprocesszor-utasítás feldolgozásának a végén következik be, a főprocesszornak rövid (de az előbbiekben említettől eltérő) veremformátumot kell választania. A kizárás vektorszámát a társprocesszor előre megadja. Ezt az a primitív válasz adja meg, ami a kizárást kiváltja.

A társprocesszor-koncepció arra is lehetőséget nyújt, hogy a társprocesszor belső állapotát a tárolóban megőrizzük, és a későbbiekben visszaállítsuk. A társprocesszor bizonyos mennyiségű byte-ot (ill. szót) az instruction-streamtől kezdődően) a programszámlálótól lekérdezhet.

4. A 68000-ES PERIFÉRIÁI

A 68000-es processzorcsalád a 6800-as sorozatnak, tehát a család 8 bites elődjének, a piacon levő valamennyi perifériáját ki tudja szolgálni. Azért, hogy a 68000-es család lehetőségeit és teljesítményét teljes mértékben ki lehessen használni, a Motorola és más gyártó cégek új, 16 bites perifériákat fejlesztettek ki, amelyeket elsősorban a 68000-es mikroprocesszor-családdal kapcsolatos alkalmazásokhoz terveztek.

Természetesen e fejezet keretében nem lehet minden egyes egység részletkérdéseire kitérni. A rendszerkezelést és az adat(táv)átvitelt támogató hardverek jelenlegi piaci helyzetéről kívántunk áttekintést adni.

4.1 Intelligens perifériaellenőrző egység (IPC)

A 68120-as és a 68121-es intelligens perifériaellenőrző egységek önálló adatcserét tesznek lehetővé a perifériaegységekkel. Ezek az egységek saját maguk is tartalmaznak egy CPU-t, amelyek (csak a 121-esnél) maszkolhatóan programozhatók.

A tulajdonképpeni adatcsere rendszerbusz-oldalról 128 byte RAM-on keresztül történik, ami mind a 68000-es CPU-ja, mint a chipen belüli CPU által írható és olvasható.

Hat szemaforregiszter gondoskodik arról, hogy ezeknek a többfelhasználós rendszerben működő elemeknek a kritikus szakaszai biztonságban legyenek. Az egység továbbá 21 párhuzamos vezetéket, egy funkció időzítőt (timert) számlálót (3 funkció, 16 bit szélességű), valamint 2 vagy 5 handshake-vezetéket bocsát a felhasználó rendelkezésére.

Mindkét egység különböző üzemi állapotba kapcsolható, amelyek során pl. mindkét 8 bites kapu adatbuszként ill. címbuszként vagy egyszerű be- vagy kimenő vezetékként használható.

A belső CPU 8 bites, amely az operandus kódot illetően kompatibilis a 6801-essel. Az is figyelemre méltó, hogy ez az egység el tudja végezni a 8 × 8 bites szorzást, ami néhány meghajtó (lemezes meghajtó, adott esetben ellenőrző számításokat végző adatátvivő meghajtó) esetén hasznos.

A 68120/68121-es sorozat egységei kétféle sebességű verzióban kerülnek a piacra, foglalatuk 40 pólusú tok.

4.2 Buszmegszakító modul (BIM)

A 68153-as egységen nem olyan, kifejezetten a processzor mellett levő megszakítóegységet kell érteni, mint amilyenre például a 80 × 86-os sorozatok processzorainak van szüksége. Mivel a 68000-esnek van egy belső prioritási logikája, amit az nIPL0...nIPL2 vezetékeken keresztül kezel, a prioritások külső megállapítására és válogatására csak bizonyos megszakítási felkéréseknél van szükség.

Ezzel szemben viszont a 68000-esnél és a hozzá kifejlesztett buszkoncepcióknál más nehézségek merülnek fel.

Például egy viszonylag intelligens egységet kell egy buszra csatlakoztatni, amely utóbbi a rácsatlakoztatott perifériáról egy bizonyos intelligenciát feltételez. Ha egy ilyen készülék megszakítási kérést küld a processzorhoz, akkor a processzor a kérés vételét követően arra vár, hogy a felkérő egység azonosítsa magát, például egy vektorszám formájában. A 68000-es az autovektor megszakításokat is feldolgozza, viszont erre a célra csak nyolc, fix vektor áll a rendelkezésre. A nem autovektorokkal viszont egy tetszőleges, éppen szabad vektort lehet használni. Ezzel a vektorral aztán a buszmaszter elvégez(tethet)i a készülék szervizrutinját.

Mivel azonban egy ilyen megszakítás-nyugtázás-ciklus időbeli lefolyása meglehetősen bonyolult, erre a célra egy ilyen buszmegszakító modult használnak.

Az egységnek összesen négy, ún. szolgája (slave) van, mindegyiken egy bemenet, amelyeken keresztül a készülékek megszakítási kérésüket bejelenthetik. Ez például a külső egység állapotregiszterének hardveroldalról meglevő állapotbitjén (pl. megszakítási kimenet) keresztül is megtörténhet.

Ezután a 68153-as a megszakítási kérelem bejelentését a helyes módon továbbítja a CPU-hoz.

Ha ez a továbbítás sikeresen megtörtént, és az ezzel bejelentett megszakítást a CPU a legmagasabb prioritásúnak minősítette, akkor ez elindít egy megszakítás-tudomásulvételi ciklust. A CPU most egy vektort vár a megszakító egységtől, ami a jelen esetben a buszmegszakító modul. Ez egyidejűleg azt is fel tudja ismerni, hogy a lehetséges különböző megszakítások melyikére kapott választ, mert a CPU a legalsó három, A1...A3 címvezetéken a várt vektor szintjét megadja.

Ha a BIM a tőle kért megszakítási szint egyikét ezeken a címvezetékeken felismeri, akkor vektorként egy előre beprogramozott szót helyez az adatbuszra, és azután megkezdődhet a készülék tulajdonképpeni kiszolgálása egy ezáltal kiválasztandó meghajtón keresztül.

A 68153-asnak 8, programozható, nyitott kollektorú kimenete van, amelyik mindegyik, mindenféle megszakítási kéréshez megszakító vezetékként használható. Az egyes bemeneteknek a 8 kimenet valamelyikéhez való hozzárendelése az egység inicializálásakor történik meg.

Az egységnek van még két, INTAL0 és INTAL1 jelű vezetéke, amelyek bizonyos időpontokban a felkérő egységgel közölni tudják, hogy a CPU akkor éppen a vektorukra vár. Ennek akkor van értelme, ha a készüléknek van saját vektora, de az eltérő hardverstruktúrák miatt a megszakítási kérelmét saját maga, közvetlenül nem tudja elküldeni a CPU-hoz.

A programozáshoz az egységnek 8 regisztere van, amelyek három címvezetéken keresztül választhatók ki. Az első négy regiszternek ellenőrzési feladatai vannak a megszakítási csatorna inicializálásához, annak eldöntéséhez, hogy a megszakítási kérelmet – miután a CPU már válaszolt rá – magának a BIM-nek kell-e törölnie. Továbbá ezek a regiszterek rendelkezhetnek egy-egy kapcsolóval a mindenkori megszakítás be-, ill. kikapcsolására. A regiszterek különböző további kapcsolókkal is rendelkeznek.

A másik négy regiszter tartalmazza azt a négy vektort, amelyek valamelyikét a megszakítás kérésekor az adatbusz rendelkezésére kell bocsátani.

4.3 Párhuzamos interfész és időzítő (timer, PI/T)

A 68230-as egy igen egyszerű, és könnyen kezelhető egység, amelynek különböző párhuzamos be- és kimeneti vezetékei, valamint egy időzítője van.

A 16 vezeték bemenetként, kimenetként és mindkét irányban is használható.

Azt, hogy az egyes vezetékeknek bemenetként vagy kimenetként kell-e funkcionálni, egy adatirány-regiszterben levő, a vezetékekre vonatkozó bit állapota adja meg. Ha a vezetéket viszont mindkét irányban felváltva kell tudni használni, akkor megfelelő előzetes programozás után az adatirány-regiszter tartalma nem kerül figyelembevételre, és a teljes csatorna irányát a meglevő négy handshake-(nyugtázó-)vezeték állapota határozza meg.

A handshake-vezetékek bizonyos körülmények között megszakítási kérést hozhatnak létre a megszakítást ellenőrző egységen, amelynek nem kell különösképpen „intelligensnek” lennie. A párhuzamos interfész és időzítő egység (PI/T) a megszakítás-nyugtázási helyzetet saját maga fel tudja dolgozni, és egy megfelelő vektort az adatbuszra helyezni. A PI/T természetesen az autovektor megszakításokat is tudja inicializálni.

Az időzítő egységnek 24 bites számlálója van, ami elé egy 5 bites osztó kapcsolható. Ezzel a két számlálóval vagy a CPU órajelét, vagy egy kívülről érkező órajelet lehet bázisórajelként feldolgozni.

A PI/T az időzítőre kérhet külön megszakítást, ami a párhuzamos kapuk kéréséhez hasonlóan vektorként vagy autovektorként dolgozható fel.

Az időzítő periodikus impulzusokat vagy szimmetrikus négyszögjeleket állít elő, de a programozott idő eltelte után egyszerű megszakítást is létre tud hozni.

4.4 Közvetlen tárhozzáférés (DMA)

A közvetlen tárhozzáférés (DMA, Direct Memory Access), azt jelenti, hogy egy perifériának lehetősége van arra, hogy az adatait a megfelelő ellenőrző egységen keresztül közvetlenül a tárba írja, illetve onnan közvetlenül olvasson adatokat anélkül, hogy eközben a CPU-t a saját programjának futásában lényegesen zavarná. Mindössze azt kell biztosítani, hogy abban az időpontban, amikor a hozzáférés a DMA-ellenőrző részéről történik, a CPU részéről ne történhessen hozzáférés. Ez vagy úgy biztosítható, hogy a DMA csatorna mindig csak akkor férhessen a tárhoz, amikor a CPU éppen a „belső ügyeit intézi", vagy pedig, hogy a DMA-ellenőrző a buszt közvetlenül a CPU-tól kérje. Erre a célra a processzornál különböző bemenő és kimenő jelek szolgálnak. Ha beérkezik egy buszvonal iránti kérés, akkor a CPU a buszt átadhatja. Ha megteszi, akkor a DMA a buszt addig birtokolhatja, míg az saját elhatározása szerint vissza nem adja.

A 68000-es perifériának csoportjában két, a lábkiosztást és általában a működést illetően kompatibilis DMA egység van, a 68440-es és a 68450-es. Ezek egymástól csak a rendelkezésre álló csatornák számában, tehát az egyidejűleg üzemeltethető adatátvivő vezetékek számában különböznek.

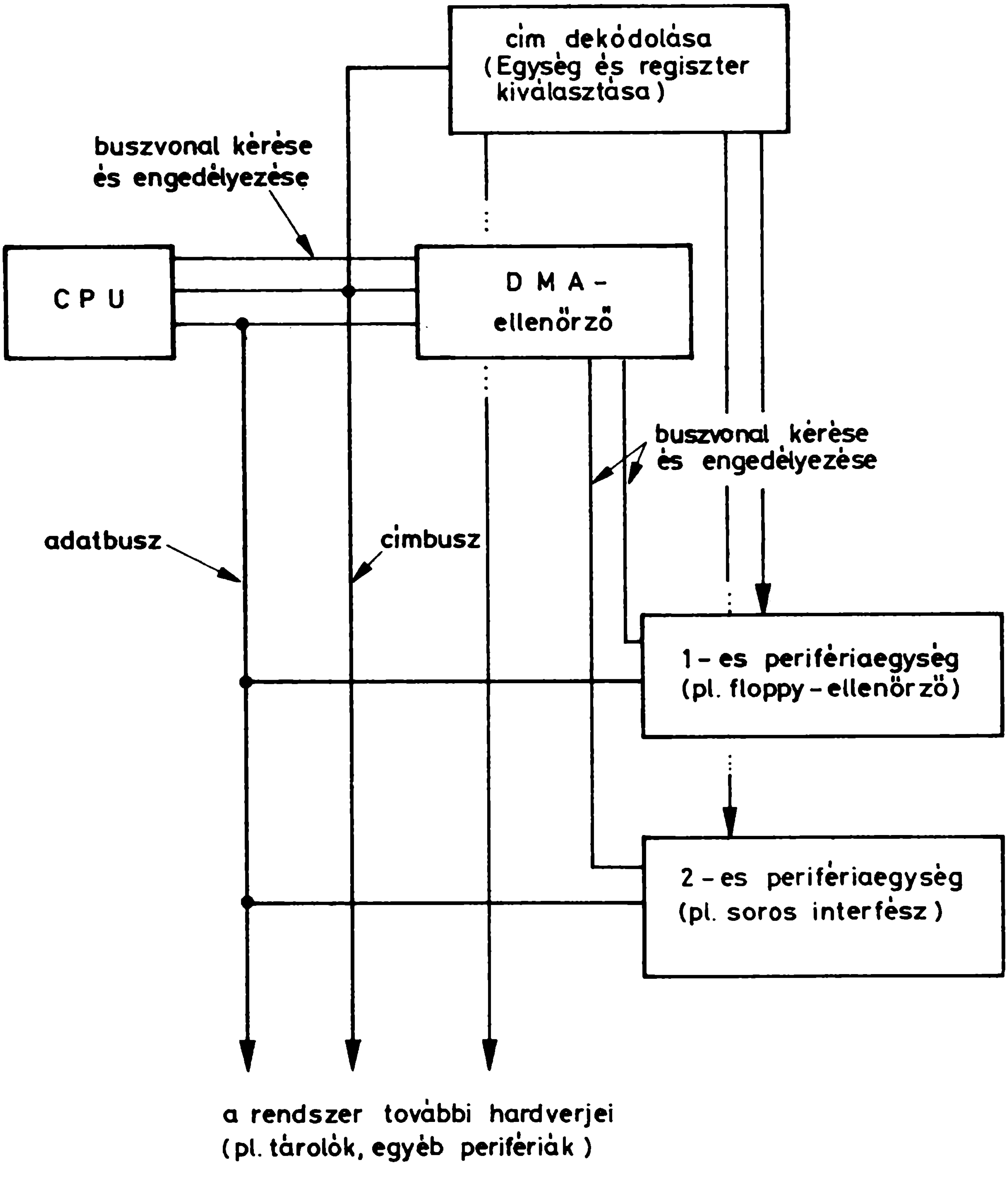
A 68440-esnek kettő, a 68450-esnek négy, egymástól független DMA csatornája van, amelyekkel adatokat lehet a perifériáról a tárba, a tárból a perifériába, ill. egyik tárból a másik tárba vinni.

Az adatátvitel maximális sebessége 4 megabyte/s, az átvihető szavak mindenkori száma a megfelelő számlálóregiszter programozásától függ.

A DMA csatornák mindegyikének van egy megszakítási vezetéke, amelyen keresztül pl. egy komplett blokkátvitel után a processzornál kérhető a program megszakítása. Ehhez mindkét egység mindegyik csatornájához két-két vektorral rendelkezik, amelyek az ellenőrző egység igénye szerint a buszra adhatók.

A buszhozzáférést valamennyi csatorna esetén vagy egy külső jel, vagy pedig egy belső regiszter kezdeményezheti. Tárból tárba való átvitelkor ezt többnyire egy regiszter kezdeményezi, mivel erre általában nem egy periféria külső felkérésekor kerül sor.

Arra a célra, hogy egy rendszeren belül egyidejűleg több DMA-ellenőrző egységet lehessen működtetni, egy másik egység, a buszkezelő egység (Bus-Arbitration-Module, BAM) szolgál.


cím dekódolása
(Egység és regiszter kiválasztása)
buszvonal kérése és engedélyezése
CPU
DMA-ellenőrző
buszvonal kérése és engedélyezése
adatbusz
címbusz
1-es perifériaegység (pl. floppy-ellenőrző)
2-es perifériaegység (pl. soros interfész)
a rendszer további hardverjei (pl. tárolók, egyéb perifériák)

DMA-ellenőrző egy mikroszámítógépes rendszerben
(a rendszerben levő tárkezelő egység nincs figyelembe véve)

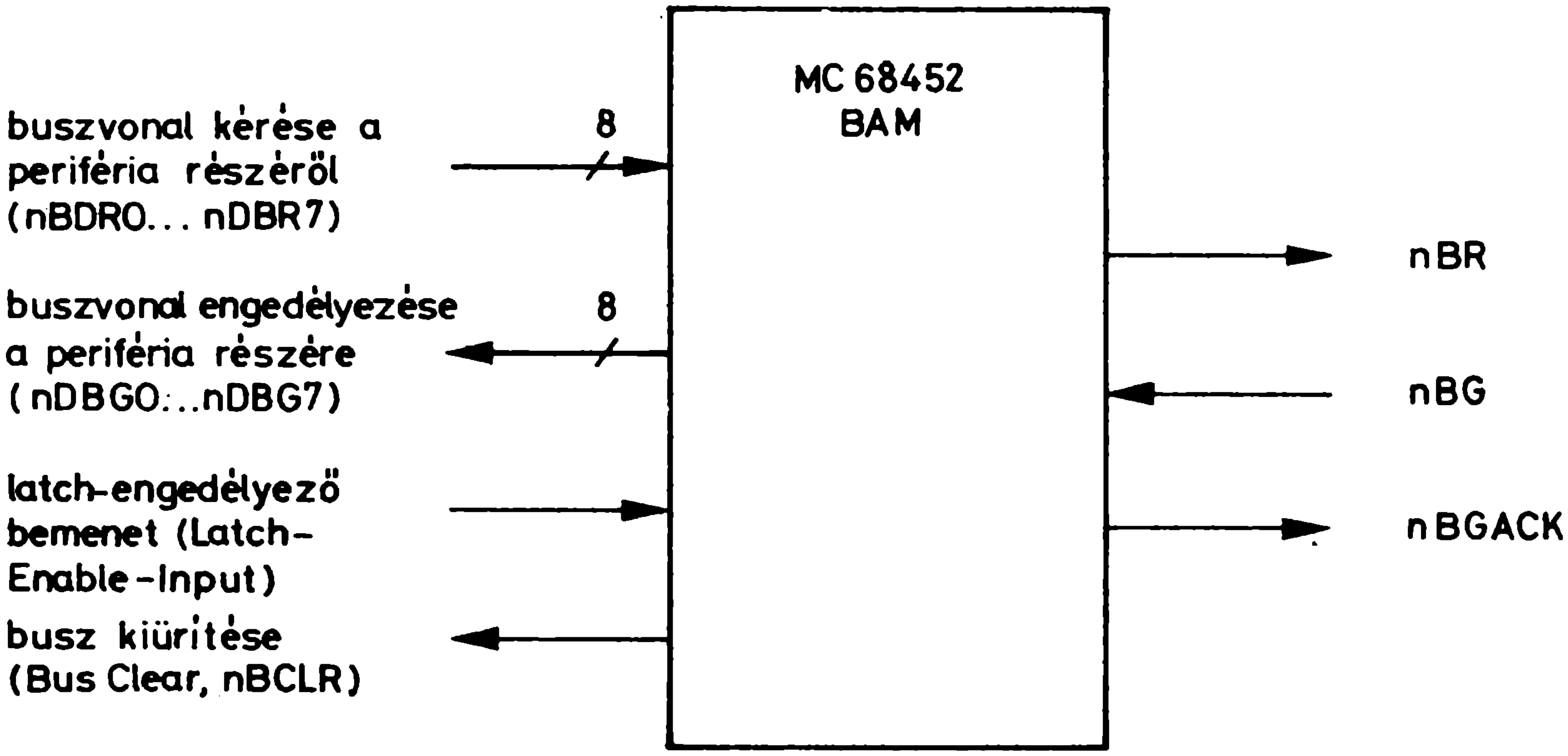
4.5 Buszkezelő egység (BAM)

A növekvő értékű rendszerek, és a manapság viszonylag olcsóbbodó intelligens perifériák egyre fontosabbá teszik, hogy olyan átfogó buszkoncepciókkal rendelkezzünk, amelyek lehetővé teszik, hogy az egyes intelligens perifériák egymást ne befolyásolják vagy zavarják. Ezzel már jó ideje a közepes és a nagy adatfeldolgozó-technika számítógépes rendszerei is foglalkoztak, ma a mikroelektronika ezeknek a rendszereknek a buszkoncepcióit részben javított formában a kisadatfeldolgozási technikában is alkalmazza.

Az ilyenkor felmerülő hardverköltségek alacsony szinten való tartása céljából fejlesztették ki a 68452-es buszkezelő egységet (Bus-Arbitration-Module, BAM), ami egységenként nyolc különböző „készülék" részére teszi lehetővé, hogy a busz masztere legyen. Ez úgy történik, hogy a felkérő készülék egy DEVICE-BUS-REQUEST (készülék-busz-kérés) jelet küld a buszkezelő egység (BAM) részére. Ez a megszakításellenőrző egységhez hasonlóan megvizsgálja a bemenet prioritását, és ettől függően vagy elkéri a buszt a CPU-tól, vagy a felkérésre egyszerűen nem válaszol.

Ha viszon a BAM a CPU-tól elkérte a buszt, és meg is kapta, akkor a busz a kérést a periféria számára a megfelelő DEVICE-BUS-GRANT jellel, a CPU felé pedig az nBGACK jellel nyugtázza. Ettől a pillanattól kezdve a felkérő készülék a buszmaszter, és az is marad mindaddig, amíg a buszt újra le nem adja. Még ha egy csatorna használata közben egy nagyobb prioritású buszkérés (BUS REQUEST) érkezne a BAM-hoz, a BAM-nak nincs jogosítványa arra, hogy a buszt az alacsonyabb prioritású készüléktől elvegye. Ez a rendszer tervezését lényegesen leegyszerűsíti, mert a buszvonal oldaláról legérzékenyebb készülékeknek nem kell a legmagasabb prioritással rendelkezniük. Más rendszereknél a prioritások fokozata korlátozó tényező, mert egy felkérés során a legmagasabb fokozatú prioritásnak először meg kell szabadulnia a benne levő adatoktól, hogy a következő hozzáférésnél ne ez ismétlődjön meg újra és újra.

A BAM alkalmas arra, hogy akár DMA perifériákat, akár több CPU-t egy buszra csatlakoztasson.


MC 68452
BAM
buszvonal kérése a periféria részéről (nBDR0... nDBR7)
8
buszvonal engedélyezése a periféria részére (nDBG0...nDBG7)
8
latch-engedélyező bemenet (Latch-Enable-Input)
busz kiürítése (Bus Clear, nBCLR)
nBR
nBG
nBGACK

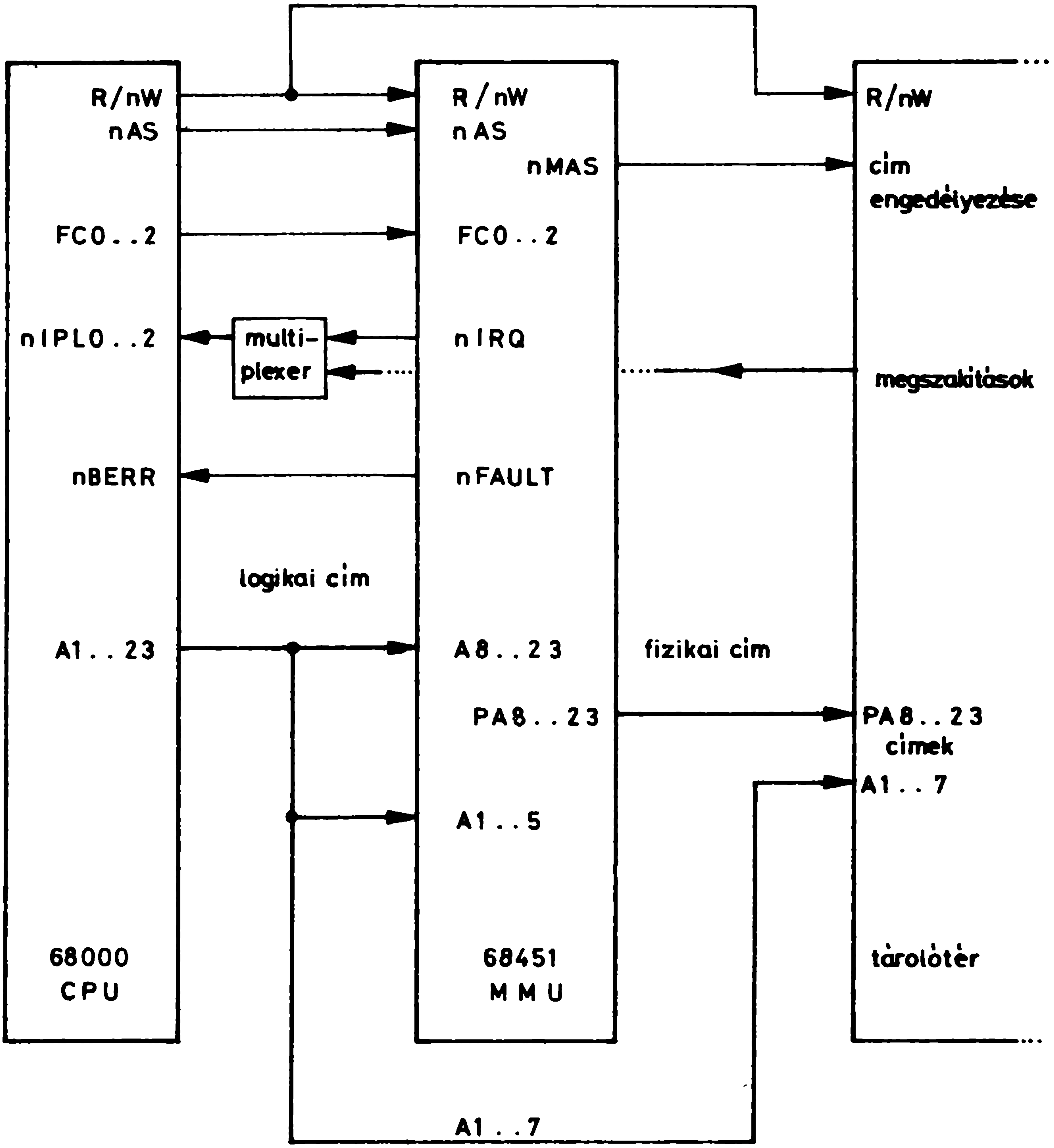
4.6 Tárkezelő egység (MMU)

A többfelhasználós rendszereknél az operációs rendszer egyik fő feladata, hogy a meglevő tárat a különböző felhasználók és az általuk igénybe vett processzorok között úgy ossza fel, hogy lehetőleg valamennyi felhasználó a részére szükséges helyet a részére szükséges címeken megkapja, és hogy az így elosztott tárterületek között átfedés ne legyen. A tárkezelés igen szerteágazó témakörét a 68000-es operációs rendszereivel foglalkozó fejezetben részletesen tárgyaljuk, ezért erre itt nem térünk ki.

A Motorola által kifejlesztett 68451-es tárkezelő egység (MMU, Memory Management Unit) a tárat egyidejűleg 32 nagy szegmensre tudja felosztani. Ha ennél több szegmensre lenne szükség, akkor a 68451-es további MMU-kkal bármikor kaszkádolható.

Az MMU a tárkezeléshez bemenő jelekként az A8...A23 címvezetékek jeleit, valamint az FC0...FC2 funkciókódot használja. Ismeretes, hogy e három vezetéknek az a feladata, hogy a tárolót USER (felhasználói) és RENDSZER-területekre, valamint továbbá program- és adattárrá ossza fel. Ez jelentős mértékben megnöveli a programok biztonságos futását és az adatok védelmét.

A 68451-es MMU mellett létezik még a 68851-es MMU is. Ez egy 68020-as rendszerben társprocesszorként használható, ezen kívül a 68851-essel a szegmensek tetszőlegesen, 4 kbyte nagyságú oldalakra bonthatók. Egy rendszerben a 68851-es MMU-nak mindig a 0! társprocesszornak kell lenni, mert ez ezt a számot saját maga dekódolja.


R/nW
nAS
FC0..2
nIPL0..2
multi-plexer
nBERR
logikai cím
A1..23
68000
CPU
R/nW
nAS
nMAS
FC0..2
nIRQ
nFAULT
A8..23
PA8..23
A1..5
68451
MMU
fizikai cím
A1..7
R/nW
cím engedélyezése
megszakítások
PA8..23
címek
A1..7
tárolótér

68000-es rendszer egy 68451-es MMU-val (elvi rajz)

4.7 Soros kimeneti-bemeneti egységek

Az adatátvitel legelterjedtebb eljárásának megvalósításához, a soros adatátvitelhez a 68000-es sorozat különböző egységei állnak rendelkezésre, amelyeknek különböző előnyeik és hátrányaik vannak.

Elsőként a 68564-est említjük, amely egy kétcsatornás, soros, szinkron és aszinkron adatátvitelre alkalmas adó-vevő egység. Minden egyes jelhez 5, 6, 7 vagy 8 bitet, valamint a szokásos 1, 1½ vagy 2 stopbitet és esetlegesen paritásbiteket tud továbbítani. A szinkron csatornákon való adatátvitelhez a bit- vagy byte-onkénti szinkronizálás különböző lehetőségeivel rendelkezik. Ezenkívül rendelkezik egy, az adatátvitel sebességét beállító generátorral (Baudrate generátorral), ami az adatátvitel sebességét egy csatlakoztatandó kvarc alapfrekvenciájának osztásával állítja be.

A 68652-es MPCC (Multi-Protocol Communications Controller) csak szinkron adatátviteli állomásokon alkalmazható, és valamennyi, ma érdekes hardverprotokollt támogatja. A 0...2 megabit/s sávszélességével egy kifejezetten gyors egység, amely számítógép és számítógép, soros csatornákon keresztüli, közvetlen összekapcsolására is alkalmas.

Az EPCI (Enhanced Programmable Communications Interface) egy, a 6800-as sorozatból ismert 2651-es PCI-k utódja. Ez egyformán jól használható soros szinkron és aszinkron adatátvitelre. Ez a 68564-es SIO-hoz (Serial Input/Output) hasonlóan rendelkezik egy, az adatátvitel sebességét beállító generátorral (Baudrategenerator), különböző módon támogatja a szinkron hardverprotokollokat, és van egy beépített auto-echo-funkciója, ami egy beérkező jelet automatikusan visszaküld.

A szinkron átviteli üzemmódban az EPCI egy 1 megabit/s átviteli sebességet ér el. Az egység háromféle változatban szállítható, amelyek egymástól csak az aszinkron adatátviteli sebességekben különböznek.

A második két adó-vevő egységhez egy ún. Polynom-Generator/Checker (PGC) csatlakoztatható. Ennek a típusjele 68635, a feladata pedig az, hogy folyamatosan ellenőrizze azt az adatátviteli vezetéket, amelyre csatlakoztatva van. Előre programozott eljárás szerint képzi az ún. ciklikus redundancia ellenőrző összegét (Cyclic Redundancy Check, CRC), amelyeket aztán a processzor elolvashat és feldolgozhat. Négy programozható szituáció esetén egy megszakítást idéz elő a PGC, hogy a processzort a kizárási szituációkról tájékoztassa.

A Dual Asynchrone Receiver/Transmitter (DUART, 68681) egy kettős adó-vevő egység, aminek viszont az előző típusokhoz képest van néhány különlegessége. Elsőként említjük meg a belső kvarcoszcillátort, ami egy Baudrategenerátort vezérel. Ennek a generátornak 18 különböző, fixen beállított sebességfokozata van 50–38 400 Baud tartományban, a szokásos lépéstávolságokban. Ezen kívül a csatornák közül egyet időzítőként lehet használni.

A vezetékeket illetően különböző hibafelismerő funkciója van, amelyek elsősorban vezetéktörések és törési feltételek (break) felismerésére vonatkoznak.

Hibaközlés, ellenőrzőjel keltés és felismerés céljára az egységben 6 bemenő és 8 kimenő vezeték van, amelyek messzemenően szabadon programozhatók.

Az EPCI adatátviteli sebesség 1 megabit/s.

4.8 Kombinált soros/párhuzamos kimeneti-bemeneti egységek (MFP)

Az egységek e csoportjába tartozik a 68901-es többfunkciós periféria (Multi-Function-Periphery, MFP), amelynek egy soros, szinkron vagy aszinkron módon működtethető adatátviteli vezetéke, négy időzítőegysége és egy 8 bit szélességű, párhuzamos kimenő-bemenő csatornája van.

Ezt az egységet főként olyan rendszerkártyákon használják, ahol a rendszert alapvetően csak egy felhasználó használja. Az MFP a négy időzítővel két késleltető időzítőt és két, szabadon programozható időzítőt bocsát a felhasználó rendelkezésére. A négy időzítő közül kettő Baudrategenerátorként használható a soros interfészhez. A soros interfész egy kvarccal vagy külső vezetékeken keresztül aszinkron módban 62 500 bit/s, szinkron üzemmódban 1 megabit/s felső sebességhatárig működtethető.

A párhuzamos kapu vezetékei megszakítási bemenetekként is használhatók. Így a 68901-es MFP nyolc külső megszakítási forrást tud kezelni. Ezen kívül van nyolc további belső megszakítási forrás. A 68901-es minden egyes megszakítási forrásnak egy saját vektorszámot tud a buszra küldeni. Ezen kívül megvannak a megfelelő vezetékek egy DMA egységen keresztül való közvetlen tárhozzáférés céljára.

4.9 Lebegőpontos aritmetika társprocesszor (FPP)

Ez az egység az eddig tárgyalt egységektől szerkezetében eltér. Míg a normál perifériaegységek a tároló vagy a kimenet/bemenet területén dolgoznak, ez a 68881-es egy olyan társprocesszor, amely teljes mértékben a 68020-as társprocesszor-koncepciójához készült. Ennek a koncepciónak a részletes leírása az azonos című fejezetben található.

Ez a társprocesszor alapvetően nyolc, 80 bit szélességű lebegőpontos adatregiszterrel, három ellenőrző regiszterrel, feltételkódokkal (flagek) és egy utasításmutatóval rendelkezik.

A 68881-es FPP a 68020-as felhasználója számára négy új adattípust kínál:

Lebegőpontos számok egyszeres pontossággal	(S)
Lebegőpontos számok kétszeres pontossággal	(D)
Lebegőpontos számok bővített pontossággal	(X)
Lebegőpontos számok tömörített BCD-láncai	(P)

Az itt bevezetett S, D, X és P betűket új operandus-deszkriptorokként kell értelmezni. Míg eddig csak a B, W és L betűket használtuk a byte, szó és kettős-szó jelölésére, az assemblernek most már ezeket az azonosító jeleket is meg kell tudni értenie.

Az FPP által elvégezhető műveletek 5 csoportba sorolhatók:

Diadikus műveletek (műveletek két operandussal)
Monadikus műveletek (műveletek egy operandussal)
Adatátvitel és -konverzió
Bizonyos feltételek vizsgálata
Ellenőrző műveletek

A diadikus műveleteknél egy operandus forrása vagy egy 68020-as adatregiszter, a 68020-as tárának egyik tárcíme, vagy egy FPP adatregiszter. A másik operandus forrásaként, amelyet céloperandusnak nevezünk, egy FPP adatregisztert használunk, és aztán a művelet eredményét ismét egy FPP adatregiszterbe helyezzük.

A monadikus műveleteknél forrásként ugyancsak egy CPU adatregisztert, a 68020-as tárának valamelyik tárcímét vagy egy FPP adatregisztert használunk. Célként megint egy FPP adatregiszter szolgál.

Adatoknak az FPP adatregiszerekből való kivételekor, vagy külső adatoknak ezek valamelyikébe való beolvasásakor gyakran van szükség típuskonverzióra. Így egy dátumnak egy CPU-regiszterből vagy a 68020 valamelyik tárcíméről való beolvasásakor a beolvasott dátum automatikusan bővített pontosságú értékké (X) kerül átalakításra, ha az már eleve nem olyan volt.

A vizsgálati műveletek az FPP-ben meglevő kapcsolókra (flagek) vonatkoznak, a 68020-as CPU-n belüli kapcsolók vizsgálatához hasonlóan.

Az ellenőrző műveletekre ahhoz van szükség, hogy az FPP belső vezérlőregisztereit programozni lehessen.

A 68881-es FPP a számábrázolást, a kerekítések, illetve a pontatlanságok levágásának módját és a számítási módokat illetően megfelel a IEEE-szabvány szerint lebegőpontos aritmetikának. Valamennyi funkció és az IEEE által megkívánt műveletek legnagyobb része a hardverbe beépített.

Az adatoknak az FPP regiszterein belüli elhelyezése a következő séma szerinti:

v	e	m
(v = előjel,	e = exponens	m = mantissza)

Az operandusok különböző hosszúságától függően oszlanak el a rendelkezésre álló bitek:

	Szélesség bitekben		
	(S)	(D)	(X)
előjel	1	1	1
exponens	8	11	15
mantissza	23	52	64
	32	64	80

A következő matematikai alapműveletek állnak rendelkezésre:
összeadás
kivonás
szorzás
osztás
maradék meghatározása
összehasonlítás
négyzetgyökvonás
egész rész meghatározása (Integer)

Kerekítésnél a következők közül lehet választani:
kerekítés a legközelebbi számra
kerekítés a legközelebbi kisebb számra
kerekítés a legközelebbi nagyobb számra
kerekítés nullával

A kerekítési pontosság általában (X), de csökkenthető (D)-re vagy (S)-re.

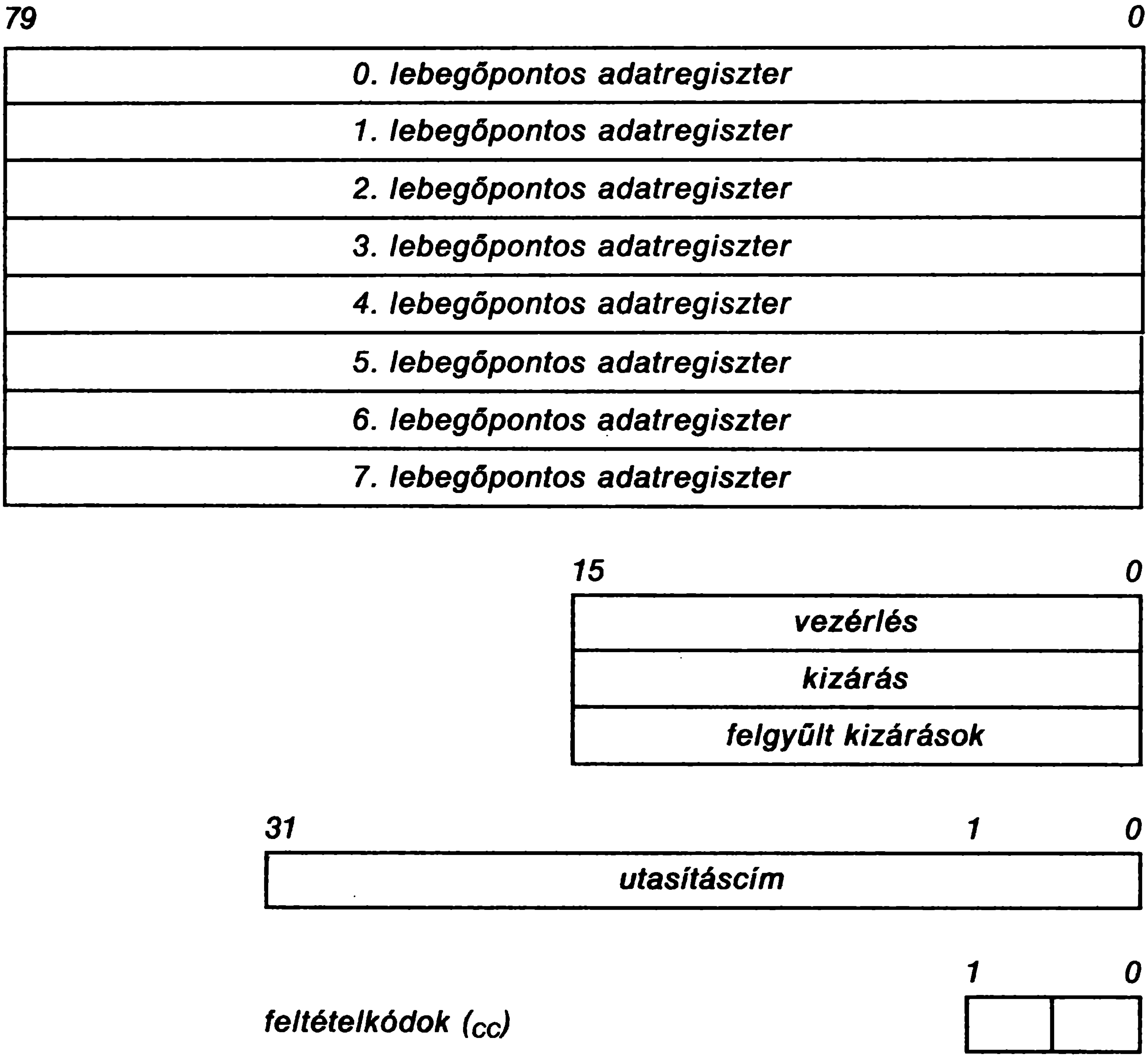
Az IEEE által megkívánt műveletek mellett a 68881-es FPP „megért” még néhány továbbit is, amely őt más, korábbi típusoktól megkülönbözteti:
abszolútérték meghatározás
negálás
egy byte beállítása a kapcsoló értékétől függően
ugrás, a kapcsolók értékétől függően
indexfeldolgozás lebegőpontos típusokra
konstansok töltése az FPP adatregiszterekben
valós szám törtjének meghatározása
valós szám exponensének meghatározása
modulo-érték feldolgozás

A 68881-es FPP-be hardveresen a következő függvények vannak beépítve:

SIN (x)
COS(x)
ARC TAN(x)
LOG DUALIS(x)
e^x
ln x

Mindegyik függvény kiszámítása automatikusan bővített pontossággal történik.

Az MC6881 lebegőpontos processzor programozási modellje


79
0
0. lebegőpontos adatregiszter
1. lebegőpontos adatregiszter
2. lebegőpontos adatregiszter
3. lebegőpontos adatregiszter
4. lebegőpontos adatregiszter
5. lebegőpontos adatregiszter
6. lebegőpontos adatregiszter
7. lebegőpontos adatregiszter
15
0
vezérlés
kizárás
felgyűlt kizárások
31
1
0
utasításcím
1
0
feltételkódok (cc)

5. AZ UTASÍTÁSOK ÉS CÍMZÉSMÓDOK

A rendszer használhatóságát illetően a processzor felépítése és a létező perifériális egységek jellemzői, rugalmassága és beszerezhetősége mellett még egy másik szempont is döntő: az utasításkészlet a rendelkezésre álló címzésmódokkal összefüggésben.

Egy rendszer használója számára az ugyanis nem nagy segítség, hogy jóllehet magának a rendszernek van ugyan egy rendszer-architektúrája, amellyel mégoly bonyolult műveleteket is el tud végezni, de a használónak nincs lehetősége arra, hogy a processzort rá tudja bírni ezeknek a regisztereknek az ésszerű alkalmazására.

Ezzel szemben az sem jó, ha igen sok utasítás között lehet válogatni, amelyek mindegyike implicit módon valamelyik regiszterre vonatkozik. Ily módon a lehetőség arra korlátozódik, hogy egy olyan, meghatározott és kompakt utasításkészletünk legyen, ami a rendelkezésünkre álló regiszterek mindegyikére vagy legalábbis a többségére alkalmazható. Magától értetődően ennek során a speciális CPU-koncepciókat is figyelembe kell venni, ahogyan ez pl. a 68020-asnál és a társprocesszoroknál (koprocesszoroknál) meg is valósult.

Az utasításkészlet tehát arról tájékoztatja a használót, hogy ezen a mikroprocesszoron a mikroprogram vagy más módszer alkalmazásával milyen elemi műveletek állnak a rendelkezésére.

A 68000-ès utasításai a következőképpen csoportosíthatók:
adatátviteli utasítások
aritmetikai utasítások
logikai utasítások
eltolási és elforgatási utasítások
bitmanipulációs utasítások
BCD (binárisan kódolt decimális) aritmetikai utasítások
programellenőrző utasítások

A 68020-asnál még a következő utasítások is vannak:
bitmezőműveletek
társprocesszor-utasítások

Ezen utasítások legtöbbje tartalmaz valamilyen formában egy vagy több *operandust*, amelyek elvileg a következő *osztályokba sorolhatók:*
bit
byte

szó
kettős-szó
BCD (tömörített)

valamint a 68020-asnál még:
bitmezők
négyes-szavak (quadwords osztásnál és szorzásnál)
nem tömörített BCD.

Ha egy utasításnál az operandus hosszúsága implicite nincs meghatározva, akkor ezt explicit módon kell elvégezni.

Ehhez a legtöbb assembler olyan bővítő mechanizmust szolgáltat, ami megadja az operandus hosszúságát. Például MOVE Dx,Dy utasítás, amely a Dx adatregiszterből adatot visz át a Dy adatregiszterbe (MOVE = átvinni).

Ebből az utasításból az azonban nem derül ki egyértelműen, hogy eközben valójában milyen hosszúságú operandust is kell feldolgozni. Az utasítás leírásából az viszont látszik, hogy az operandus hosszúsága lehet akár byte, akár szó, vagy akár kettős-szó is.

Az utasítás helyes szintaxisa:

```
MOVE.X   Dx,Dy
```

ahol az X helyére az operandus hosszúságának megfelelő rövidítést kell írni, azaz

B betűt, ha byte hosszúságú (Byte = 8 bit)
W betűt, ha szó hosszúságú (Word = 16 bit)
L betűt, ha kettős-szó hosszúságú (Longword = 32 bit)

operandussal van dolgunk.

Ha nem adjuk meg az operandus hosszúságát, akkor ezt a legtöbb assembler alapértelmezésben, azaz szó hosszúságban (16 bit) dolgozza fel.

Ha tehát a példánkban egy 32 bites átvitelt akarunk elvégezni, akkor a helyes utasítás így néz ki:

```
MOVE.L   Dx,Dy
```

5.1 Címzésmódok

A operandusok hosszúsága mellett legalább ennyire fontos, hogy mely operandusokat használjunk. A 68000-esnél semmiféle technikai vagy logikai megfontolás sem indokolja, hogy minden egyes utasítást a rendelkezésre álló valamennyi címzésmóddal lehessen használni.

Ennek egy jellemző példajaként nézzük meg, hogyan történik a veremmutató (SP = Stack Pointer) kezelése. Normál esetben a MOVE utasításnál olyan címzésmód használható, ami azt mondja meg, hogy a megadott operandust oda kell elhelyezni, ahová egy ugyancsak megadandó regiszter mutat, amelyet az információ elhelyezése előtt, az operandus byte-okban számított mindenkori hosszúságával csökkenteni kell.

Ha a művelet

```
MOVE.B  D3,-(A0)
```

alakú, akkor a 3-as számú adatregiszter alsó 8 bitje a nullás számú címregiszter által képzett verembe kerül, ezután az A0 regiszter dekremendálódik.

Ha az utasítás

```
MOVE.B  D3,-(A7)
```

alakú, akkor első látásra úgy tűnik, hogy megegyezik a fentivel. Ha viszont figyelembe vesszük azt, hogy az A7-es regiszter tulajdonképpen programellenőrző veremmutatóként (stack pointer) szolgál, akkor világossá válik, hogy ebben a veremben mindig csak szó vagy kettős-szó hosszúságú információk lehetnek, úgy hogy a processzor ide egy komplett szót helyez el.

A 68000-es processzornak az utasításszavain belül adott esetben van olyan területe, amelyben a címzésmód egy regiszter egy esetleges specifikációjával együtt adott. Ezt a kétszer három bitből álló szerkezetet effektív címnek nevezzük, és a normál alakja:

módus			*regiszter*		
b5	b4	b3	b2	b1	b0

Ha egy utasításban két effektív cím van (pl. MOVE), akkor a két cím elrendezése szimmetrikus:

regiszter			módus			módus			regiszter		
b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0

Ahhoz, hogy bármely műveletnél eldönthessük, melyik címzésmód a megengedett, beszéljük meg először valamennyi lehetségest.

A regiszter közvetlen címzése

Ennél a címzésmódnál az operandust vagy beírjuk, vagy kiolvassuk a megadott regiszterből. Regiszterként a 8 adatregiszter, a 8 címregiszter, vagy egy, vagy több ellenőrző regiszter, mint az SR, CCR, illetve a csak a 68010-esnél és a 68020-asnál meglevő VBR, SFC, DFC, CACR adhatók meg.

Itt figyelembe kell venni azt, hogy bizonyos körülmények között, meghatározott regiszterek használatakor egy utasítás valamilyen speciális utasítássá válhat, ha pl. az állapotregiszter SUPERVISOR részébe kell írni.

Adatregiszter közvetlen címzése

E címzésmód általános szintaxisa:

Dn

a módus 000. Az adatregiszterek száma 0 és 7 között lehet, ennek megfelelően a regiszterek jelölése D0-tól D7-ig terjed.

Például:

effektív cím = 000 011
a D3 adatregiszterbe írunk vagy onnan olvasunk.

Címregiszter közvetlen címzése

E címzésmód általános szintaxisa:

An

a módus 001. A címregiszter száma 0 és 7 között lehet, ennek megfelelően a regiszterek jelölése A0-tól A7-ig terjed.

Például:

effektív cím = 001 100
az A4 címregiszterbe írunk vagy onnan olvasunk.

Regiszter közvetett címzése

Ennél a címzésmódnál egy címregisztert egy, a tulajdonképpeni operandusra mutató regiszterként használunk. Ha az operandus nem egy byte hosszúságú, akkor a megadott címregiszter az operandus legalsó 8 byte-jára mutat.

E címzésmód általános szintaxisa:

(An)

a módus 010. A címregiszterek száma 0 és 7 között lehet, ennek megfelelően a regiszterek jelölése A0-tól A7-ig terjed.

Például:

effektív cím = 010 100
A cím kiszámításához az A4 címregisztert használjuk. Tegyük fel, hogy az A4-ben a $80001234 érték van, akkor az operandus legalsó byte-ja a $80001234 címen áll.

Regiszter közvetett címzése utólagos inkrementálással

E címzésmódnál egy címregisztert egy, a tulajdonképpeni operandusra mutató regiszterként használunk. Ha az operandus nem egy byte hosszúságú, akkor a megadott címregiszter az operandus legalsó 8 byte-jára mutat. Ha elővesszük ezt az operandust, akkor ezt követően a mutató az operandus byte-okban számított hosszával megnövekszik (inkrementálódik).
E címzésmód általános szintaxisa:

(An)+

a módus 011. A címregiszterek száma 0 és 7 között lehet, ennek megfelelően a regiszterek jelölése A0-tól A7-ig terjed.

Például:

effektív cím = 011 100
A cím kiszámításához az A4 címregisztert használjuk. Tegyük fel, hogy az A4-ben a $80001234 érték van, akkor az operandus legalsó byte-ja a $80001234 címen áll. A hozzáférés után a címregiszter az operandus utáni byte-ra mutat. Ha a $80001234 címen egy kettős-szó van, akkor a művelet elvégzése után az A4 tartalma $80001234, ha egy szó van, akkor $80001236, ha pedig egy byte, akkor $80001235 lesz.
Ha címregiszterként az A7-et (USP-ként vagy SSP-ként) adjuk meg, akkor a byte-operandusokból automatikusan szóoperandusok lesznek.

Regiszter közvetett címzése előzetes dekrementálással

E címzésmódnál egy címregisztert egy, a tulajdonképpen operandusra mutató regiszterként használunk. A megadott címregiszter a tárolóban az operandus fölött levő byte-ra mutat. Az operandushoz való hozzáférés előtt a mutató (regiszter) az operandus byte-okban számított hosszával dekrementálásra kerül.

E címzésmód általános sztaxisa:

–(An)

a módus 100. A címregiszter száma 0 és 7 között lehet, ennek megfelelően a regiszterek jelölése A0-tól A7-ig terjed.

Például:

effektív cím = 100 100
A cím kiszámításához az A4 címregisztert használjuk. Tegyük fel, hogy az A4-ben a $80001234 érték van, akkor az operandus legfelső byte-ja a $80001233 címen áll. A hozzáférés után a címregiszter az operandus legalsó byte-jára mutat. Amennyiben a $80001234 címre történő hozzáférés kettősszó hosszúságú, akkor a hozzáférés végrehajtása után az A4 regiszter értéke $80001230, szó hosszúság esetén $80001232 byte hosszúság esetén pedig $80001233 lesz.

Ha címregiszterként az (USP vagy az SSP) A7 regisztere kerül megadásra, akkor a byte szélességű operandusok automatikusan szó szélességű operandusokká bővülnek.

Regiszter közvetett címzése címkülönbséggel

E címzésmódnál egy címregiszter és egy 16 bites konstans érték összege mutat a tulajdonképpen operandusra. A képzett összeg az operandus legalsó byte-jára mutat. A 16 bites címkülönbség automatikusan, előjelhelyesen 32 bitre bővül, mielőtt a cím kiszámítása megtörténne.

E címzésmód általános szintaxisa:

d_{16} (An)

a módus 101. A címregiszter száma 0 és 7 között lehet, ennek megfelelően a regiszterek jelölése A0-tól A7-ig terjed.

Például:

effektív cím = 101 100
A cím kiszámításához az A4 címregisztert használjuk. Tegyük fel, hogy az A4-ben a $80001234 érték van, és a 16 bites címkülönbség $4444, akkor az operandus legalsó byte-ja a

$80001234
+$00004444
$80005678

címen áll.

Regiszter közvetett címzése címkülönbséggel és indexszel

Ennél a címzésmódnál egy címregiszter, egy 8 bites konstans, valamint egy címregiszterben levő 32 bites index mutat a tulajdonképpeni operandusra. Az előzőekből képzett összeg az operandus legalsó byte-jára mutat. A 8 bites címkülönbség automatikusan, előjelhelyesen 32 bitre bővül, mielőtt a cím kiszámítása megtörténne. Indexregiszterként bármelyik adat- vagy címregiszter használható.

E címzésmód általános szintaxisa:

d_8 (An, R.X)

a módus 110. A címregiszter száma 0 és 7 között lehet, ennek megfelelően a regiszterek jelölése A0-tól A7-ig terjed. Az Rn az indexregisztert jelöli.

Például:

effektív cím = 101 100
A cím kiszámításához az A4 regisztert használjuk. Tegyük fel, hogy az A4-ben a $80001234 érték van, az indexregiszter a D6, amelynek a 32 bites értéke $12345678, és a 8 bites címkülönbség $44, akkor az operandus legalsó byte-ja a

$80001234
+$12345678
+$00000044
$923468F0

címen áll.

Regiszter közvetett címzése indexszel

Ez a címzésmód a 68020-as sajátossága. Ennél a címzésmódnál egy címregiszter tartalmának, egy 16 vagy 32 bites konstans (bd, base-data = bázisadat) értékének, valamint – valamelyik regiszterben levő, 32 bit széles indexű és az operandus byte-ban számított hosszúsága szorzatának – az összege mutat a tulajdonképpeni operandusra. Az így képzett összeg az operandus legalsó byte-jára mutat. A 16 vagy 32 bit címkülönbség adott esetben előjelhelyesen 32 bite bővül, mielőtt a cím kiszámítása megtörténne. Indexregiszterként bármelyik adat- vagy címregiszter használható.

E címzésmód általános szintaxisa:

bd(An,Rn.X*faktor)

a módus 110. A címregiszter száma 0 és 7 között lehet, ennek megfelelően a regiszterek jelölése A0-tól A7-ig terjed. Az Rn az indexregisztert jelöli.

Például:

effektív cím = 110 100
A cím kiszámításához az A4 címregisztert használjuk. Tegyük fel, hogy az A4-ben a $80001234 érték van, az indexregiszter a D6, amelynek a 32 bites értéke $12345678, a faktora 2, és a 32 bites címkülönbsége $11112222, akkor az operandus legalsó byte-ja a

$$\begin{array}{rr} & \$80001234 \\ +\$12345678*2 = & +\$2468ACFO \\ & +\$11112222 \\ \hline & \$B579E146 \end{array}$$

címen áll.

Ha ennél a címzésmódnál egy adatregiszter mint indexregiszter mellett címregiszter és címkülönbség nincs megadva, akkor az adatregiszter közvetett címzése elnevezésű címzésmód kerül szimulálásra, amire a fejlesztők korábban egyébként nem gondoltak.

Tároló közvetett címzése utólagos indexszel

Ezt a címzési módot a 68020-as számára tartották fenn. A cím kiszámítása több lépésben történik:

Először a megadott címregiszter, és a megadott 16/32 bit szélességű, basedata („bd”, bázisadat) értékeinek előjelhelyes összeadása történik meg. Ez az összeg egy, a tárolóban levő olyan kettős-szó legalacsonyabb byte-jára mutat, ami most került betöltésre. Ehhez az értékhez a tárolóban hozzáadódik még az indexregiszter értékének és az operandus megadott hosszának a szorzatából adódó érték, és végül még az ún. szélső címkülönbség (od = outer displacement).

Az így képzett cím az operandus legalsó byte-ja. A 16 vagy 32 bites címkülönbségek adott esetben előjelhelyesen 32 bitre bővülnek, mielőtt a cím kiszámítása megtörténne. Indexregiszterként bármelyik adat- vagy címregiszter használható.

E címzésmód általános szintaxisa:

([bd, An] R.X* faktor, od)

módus 110. A címregiszter száma 0 és 7 között lehet, ennek megfelelően a regiszterek jelölése A0-tól A7-ig terjed. Az Rn az adatregisztert jelöli.

Például:

effektív cím = 110 100
A cím kiszámításához az A4 címregisztert használjuk. Tegyük fel, hogy az A4-ben a $80001234 érték van, és a bd (base-data, bázisadat) értéke $11112222, akkor a közvetett tárolóértéke legalsó byte-ja a

```
  $80001234
+ $11112222
-----------
  $91113456
```

címen áll.

Ha most mondjuk a $91113456 címen a $76543210 érték van, és az index adatregisztere a D6, amelynek az értéke $10000001 és a faktora 2, valamint az od (outer displacement = szélső címkülönbség) értéke $0005, akkor a következők szerint kerül a cím kiszámításra:

```
                     $76543210
+$10000001*2 =      +$20000002
                    +$00000005
                    ----------
                    +$96543217
```

Az eredmény tehát az, hogy a kívánt operandus legalsó byte-ja ezen a $96543217 címen van.

Tároló közvetett címzése előzetes indexszel

Ezt a címzési módot a 68020-as számára tartották fenn. A cím kiszámítása több lépésben történik:

Először a megadott címregiszter és a megadott 16/32 bit szélességű bd (base-data, bázisadat) értéke, valamint az indexregiszter értékének és az operandus hosszának a szorzata kerül előjelhelyesen összeadásra. Ez egy olyan a tárolóban levő kettős-szó legalsó byte-jára mutat, amely most került betöltésre. Ehhez az értékhez a tárolóból még egy ún. od (outer displacement, szélső címkülönbség) kerül hozzásadásra.

Az így képzett cím az operandus legalsó byte-ja. A 16 vagy 32 bites címkülönbségek adott esetben előjelhelyesen 32 bitre bővülnek, mielőtt a cím kiszámítása megtörténne. Indexregiszterként bármelyik adat- vagy címregiszter használható.

E címzésmód általános szintaxisa:

([bd, An, Rn.X*faktor] od)

a módus 110. A címregiszter száma 0 és 7 között lehet, ennek megfelelően a regiszterek jelölése A0-tól A7-ig terjed. Az Rn az indexregisztert jelöli.

Például:

effektív cím = 110 100
A cím kiszámításához az A4 regisztert használjuk. Tegyük fel, hogy az A4-ben a \$80001234 érték van, legyen az index adatregiszter a D6, ennek értéke \$10000001, a faktor 2, a bd értéke \$11112222, akkor a közvetett tárolóérték legalsó byte-ja a

$$\begin{array}{lr} & \$80001234 \\ +\$10000001*2 = & +\$11112222 \\ & +\$20000002 \\ \hline & \$B1113458 \end{array}$$

címen állna.

Ha most a \$B1113458 tárcímen a \$76543210 érték lenne, és az od \$0005 lenne, akkor a cím kiszámítása a következők szerint folytatódna:

$$\begin{array}{r} \$76543210 \\ +\$00000005 \\ \hline \$76543215 \end{array}$$

Ezen a \$76543215 címen van most a kívánt operandus legalsó byte-ja.

Programszámláló relatív címzése

Ennél a címzésmódnál a PC (Program Counter) programszámlálóban levő értékhez megadott, 32 bitre bővített 16 bites címkülönbség előjelhelyesen hozzáadódik. Az így képzett cím a CPU programtartományában levő operandus legalsó byte-ja.

E címzésmód általános szintaxisa:

d_{16}(PC)

A módus 111, a regiszter mindig 010.

Például:

effektív cím = 111 010
Tegyük fel, hogy a PC-ben a cím kiszámítása időpontjában a \$80001234 érték

van, és a címeltolás (displacement) értéke $1111, akkor az operandus legalsó byte-ja a

```
 $80001234
+$00001111
 $80002345
```

címen van.

A „cím kiszámításának időpontja” nagyon egyszerűen meghatározható: nulla lenne az eltolás (offszet) akkor, ha az eltolás (offszet) maga az operandus lenne:

```
MOVE.B 0(PC),D3
```

a tárban a következő értékeket eredményezné:

```
$80001232:          $163A 0000
```

ezzel szemben a

```
MOVE.W              címke(PC),D3
JMP                 el innen
címke:  .DATA       HALLO
        .EVEN
el innen: ...
```

Utasitássorozat a következő kódokat adja:

```
80001232:           163A 0006
80001236:           4AF8 1240
8000123A:           48 41 4C 4C 4F 00
80001240:           ...
```

Amint a példa mutatja, a kódrészben az ilyen adatoknál mindig ügyelni kell arra, hogy a processzor az operandus-kódjánál mindig páros címre menjen. A legtöbb assembler valamilyen formában ad ehhez egy EVEN pszeudo-műveletet, ami ennek az elvégzésére szolgál.

Programszámláló közvetett címzése indexszel

E címzésmód során a PC programszámlálóban levő értékhez hozzáadódik előjelhelyesen a megadott, 32 bitre bővített 8 bites címkülönbség, és az indexregiszter tartalma az operandus hosszúságával. Az így képzett cím a CPU programtárban levő operandus legalsó byte-ja. A 68020-as processzornál az indexregiszter értéke még egy faktorral megszorozható.

E címzésmód általános szintaxisa:

d_8 (PC) (Xn.X)

A módus 111, a regiszter mindig 011.

Például:

effektív cím = 111 011
Tegyük fel, hogy a PC-ben a cím kiszámításának időpontjában az $80001234 érték van, az indexregiszterként használt, adott adat- vagy címregiszter tartalma $32103210, és az eltolás értéke $11, akkor az operandus legalsó byte-ja a

```
 $80001234
+$00000011
+$32103210
----------
 $B2104455
```

címen van.

A „cím kiszámításának időpontja" igen egyszerűen meghatározható: nulla értékű eltolás (offszet) akkor lenne, ha az eltolás maga az operandus lenne:

```
MOVE.B 0(PC,D0),D3
```

a tárolóban az alábbi értékeket eredményezné:

```
80001232:  163B 000
```

viszont a

```
            MOVE.W   címke (PC,D0),D3
            JMP      el innen
címke:      .DATA    0001, 0002, 0003, 0004
            .EVEN
el innen: ...
```

utasítássorozat a következő kódokat állítaná elő:

```
80001232:             11FA 0006
80001236:             4EF8 1242
8000123A:             0001 0002
8000123E:             0003 0004
80001242:             ...
```

A kódrészben az ilyen adatoknál mindig ügyelni kell arra, hogy a processzor az OP-kódjánál mindig páros címekre menjen. A legtöbb assembler valamilyen formában ad ehhez egy EVEN pszeudo-műveletet, ami éppen erről gondoskodik.

Programszámláló közvetett címzése 32 bites indexszel

Ez a címzésmód a 68020-as processzor számára van fenntartva. Itt a programtároló pillanatnyi értékének és egy 16 vagy 32 bit széles állandó érték (base data) összegét, valamint valamelyik regiszterben levő 32 bites index és az operandus byte-okban számított hosszának szorzatából képzett összeget mutatóként használjuk, amely a tényleges operandusra mutat. Az így képzett összeg az operandus legalsó byte-jára mutat. A 16 vagy 32 bites címkülönbség adott esetben előjelhelyesen 32 bitre bővül, még mielőtt a cím kiszámítása megtörténne. Indexregiszterként bármelyik adat- vagy címregiszter használható.

E címzésmód általános szintaxisa:

bd(PC,RN.X*faktor)

A módus 111, a regiszter száma mindig 011. Az Rn az indexregisztert jelöli.

Például:

effektív cím = 111 011
Tegyük fel, hogy a programszámláló értéke $80001234, az indexregiszter legyen a D6, az értéke a 32 bites $12345678, és a faktor 2, és a 32 bites címkülönbség legyen $11112222, akkor az operandus legalsó byte-ja a

```
                      $80001234
+$12345678*2 =  +$2468ACF0
                     +$11112222
                    ___________
                      $B579E146
```

Ha a programszámláló értékeként 0-át kell megadni, akkor a legtöbb assembler a PC helyett a ZPC (Zero-PC) szintaxist használja. Így a program tártartományában könnyen megadhatók abszolút címek anélkül, hogy a fő tárolóban levő, „saját" feladat pillanatnyi pozíciójára ügyelni kellene. Ha ennél a címzésmódnál indexregiszterként egy adatregiszter mellett címregisztert és címkülönbséget nem adunk meg, akkor ezzel a program tartományában az **adatregiszter közvetett címzése** címzésmódot használjuk, ami egyébként nem lett betervezve.

Programszámláló közvetett címzése utólagos indexszel

Ez a címzésmód a 68020-as processzor számára van fenntartva. Ennek során a cím kiszámítása több lépésben történik:

Először a PC programszámláló pillanatnyi értéke és a megadott 16/32 bites bd (bázisadat) értéke kerül előjelhelyesen összeadásra. Ez az érték egy olyan, a tárolóban levő kettős-szó legalsó byte-jára mutat, ami most került betöltésre.

A tárolóból így származó értékhez kerül hozzáadásra az indexregiszter értékének és a megadott operandushosszúság szorzatának az összege. Végül hozzáadódik még az ún. szélső címkülönbség (outer displacement, od).

Az így képzett cím lesz a operandus legalsó byte-ja. A 16 vagy 32 bites címkülönbség adott esetben előjelhelyesen 32 bitre bővül, még a cím kiszámítása előtt. Indexregiszterként bármely adat- vagy címregiszter használható.

E címzésmód általános szintaxisa:

([bd,PC] RN.X*faktor, od)

A módus 111. A regiszter száma 011. Az Rn az adatregisztert jelöli.

Például:

effektív cím = 111 011
Tegyük fel, hogy a programszámlálóban az értéke $80001234, a base-data értéke $11112222, akkor a közvetett tárolóérték legasó byte-ja a

```
  $80001234
 +$11112222
 ----------
  $91113456
```

címen áll.

Ha most a $91113456 tárcímen mondjuk a $76543210 érték van, az indexregiszter a D6, amelynek tartalma $10000001, és a 16/32 bites szélső címkülönbség (od) $0005, akkor a cím kiszámítása így folytatódik:

```
                    $76543210
+$10000001*2 =     +$20000002
                   +$00000005
                   ----------
                    $96543217
```

Ezen a $96543217 címen áll most a kívánt operandus legalsó byte-ja.

Ha a programszámláló értékeként 0-át kell megadni, akkor a legtöbb assembler a PC helyett a ZPC (Zero-PC) szintaxist használja. Így a program tártartományában könnyen megadhatók abszolút címek anélkül, hogy a fő tárolóban levő „saját” feladat pillanatnyi pozíciójára ügyelni kellene.

Programszámláló követett címzése előzetes indexszel

Ez a címzési mód a 68020-as processzor számára van fenntartva. A cím előállítása több lépésben történik:

Először a programszámláló pillanatnyi értéke és a megadott 16/32 bites base-data (bd), valamint az indexregiszter értékének és az operandus hosszúságának a szorzata kerül összeadásra, előjelhelyesen. Ez az érték egy, a tárolóban levő olyan kettős-szó legalsó byte-jára mutat, ami most kerül betöltésre. Ehhez a tárolóból származó értékhez adódik hozzá most az ún. szélső címkülönbség (outer displacement, od).

Az így kiszámított cím az operandus legalsó byte-jának a címe. A 16 vagy 32 bites címkülönbségek adott esetben 32 bitre bővülnek, még mielőtt a cím kiszámítása megtörténne. Indexregiszterként bármely adat- vagy címregiszter használható.

E címzésmód általános szintaxisa:

([bd,PC,Rn.X*faktor] od)

A módus 111. A regiszter száma 011. Az Rn az indexregisztert jelöli.

Például:

effektív cím = 111 011
Tegyük fel, hogy a programszámlálóban az érték $80001234, legyen a D6 az indexregiszter, az értéke $10000001, a faktor 2 és a base-data legyen mondjuk $11112222, akkor a közvetett tárolóérték legalsó byte-ja a

```
                      $80001234
+$10000001*2 =   +$11112222
                     +$20000002
                     ----------
                      $B1113458
```

címen áll.

Ha most a $B1113458 tárolócímen a $76543210 érték áll, és a 16/32-es szélsú címkülönbség (outer displacement, od) $0005, akkor a cím kiszámítása így folytatódik:

```
 $76543210
+$00000005
----------
 $76543215
```

Ezek a $76543215 címen áll a kívánt operandus legalsó byte-ja.

Abszolút címzés szó hosszúságú címmel

A címet egy szó adja meg, amely előjelhelyesen 32 bitre bővül, és ezt az operandus legalsó byte-jának közvetett címeként kell értelmezni.

E címzésmód általános szintaxisa:

xxxx.W

A módus 111, a regiszter száma 000.

Például:

effektív cím = 111 000
Tegyük fel, hogy a programszámláló utáni szó értéke $1234, akkor a $00001234 lenne az a cím, amely az operandus legalsó byte-jára mutat.

Abszolút címzés kettős-szó hosszúságú címmel

A címet egy kettős-szó adja meg, amelyet az operandus legalsó byte-jának közvetlen címeként kell értelmezni.

E címzésmód általános szintaxisa:

xxxx.L

A módus 111, a regiszter száma 001.

Például:

effektív cím = 111 001
Tegyük fel, hogy a programszámláló utáni szó értéke $00001234, akkor ez lenne az a cím, amely az operandus legalsó byte-jára mutat.

Adatok közvetlen címzése (konstans érték)

Ennél a címzésmódnál az operandus követlenül a megfelelő műveleti kód után adott. Ha az operandus byte hosszúságú, akkor ez a byte az utasítást követő szó alsó 8 bitjén helyezkedik el. Egy szó hosszúságú operandus ennek megfelelően az utasítást követő teljes szó-hosszúságot, egy kettős-szó hosszúságú operandus pedig a műveleti kódot követő két szót foglal el, mégpedig úgy, hogy a magasabb helyiértékű szó az alacsonyabb tárcímen helyezkedik el.

E címzésmód általános szintaxisa:

#xxxx

A módus 111, a regiszter száma 100.

Például:

effektív cím = 111 100
Tegyük fel, hogy a programszámláló utáni szó értéke $1234, akkor ez lenne a szóoperandus.

Mivel az effektív címek normál, valamennyi processzorra érvényes felépítését a fejezet elején bemutattuk, itt még megadjuk a csak 68020-asra vonatkozók szerkezetét. Kétféle formátum van:

A standard formátum

bit	15	14 13 12	11	10 9	8	7 6 5 4 3 2 1 0
	DA	Regiszter	WL	Faktr	0	Címkülönbség

és a komplett formátum:

<table>
<tr><td>bit</td><td>15</td><td>14 13 12</td><td>11</td><td>10 9</td><td>8</td><td>7</td><td>6</td><td>5 4</td><td>3</td><td>2 1 0</td></tr>
<tr><td></td><td>DA</td><td>Regiszter</td><td>WL</td><td>Faktr</td><td>1</td><td>BS</td><td>IS</td><td>BD–GR</td><td>0</td><td>I/IS</td></tr>
<tr><td></td><td colspan="10">BÁZIS címkülönbség (0, 1 vagy 2 szó)</td></tr>
<tr><td></td><td colspan="10">(Outer Displacement) 0–2 szó</td></tr>
</table>

Az egyes mezők jelentései:

DA	eldönti, hogy a következő indexregiszter adatregiszter vagy címregiszter-e: 0 = adatregiszter 1 = címregiszter
Regiszter	Itt áll a DA-val specifikált cím- vagy adatregiszter száma.
WL	megállapítja, hogy az index szó vagy kettős-szó-e 0 = szó index 1 = kettős-szó index

Faktr	meghatározza azt a faktort, amellyel az indexregiszter értékét meg kell szorozni, mielőtt még ezt az értéket a cím kiszámításához használnánk: 00 = faktor nagysága 1 01 = faktor nagysága 2 10 = faktor nagysága 4 11 = faktor nagysága 8
BS	megállapítja, hogy van-e bázisregiszter definiálva. Ha nincs, akkor a cím kiszámítása gyorsabban történik, a 0 érték figyelmen kívül hagyása következtében. 0 = bázisregiszter hozzáadódik 1 = bázisregiszter-hozzáadás elmarad
IS	Ez a bit azt határozza meg, hogy figyelembe kell-e venni az indexregiszter értékét. Tekintettel arra, hogy az index értékének kiszámítása ugyancsak időigényes, ha e bit magasra állításával nem adunk meg indexregisztert, akkor jelentős számítási időt takaríthatunk meg. 0 = indexszámításokat elvégezni 1 = indexszámításokat figyelmen kívül hagyni
BD–GR.	A bázisadat nagyságát [base-displacement size, ill. a könyv egyes helyein base-data-nak nevezi – Ford. megjegyzése] ezzel a két bittel lehet meghatározni: 00 = illegális érték 01 = nincs érték 10 = szó hosszúságú címkülönbség 11 = kettős-szó hosszúságú címkülönbség
I/IS	Ezek a bitek az IS bittel együtt kerülnek felhasználásra, és a címzésmódok pontosabb azonosítására szolgálnak arra vonatkozóan, hogy a címzés módja közvetett vagy közvetlen, ill. hogy kell-e indexelni avagy nem. IS I/IS 0 000 nincs közvetett címzés 0 001 közvetett, előindexelt, címkülönbség 0 0 010 közvetett, előindexelt, címkülönbség szó hosszúság 0 011 közvetett, előindexelt, címkülönbség kettős-szó hosszúság 0 100 foglalt 0 101 közvetett, utóindexelt, címkülönbség 0 0 110 közvetett, utóindexelt, címkülönbség szó hosszúság 0 111 közvetett, utóindexelt, címkülönbség kettős-szó hosszúság 1 000 nincs közvetett tárcímzés

1	001	közvetett tárcímzés, címkülönbség 0
1	010	közvetett tárcímzés címkülönbség szó hosszúság
1	011	közvetett tárcímzés, címkülönbség kettős-szó hosszúság
1	1xx	foglalt

Az egyes címzésmódok egyszerűbb kategorizálásához ezek négy csoportba vannak osztva:

Az **adatok** kategória valamennyi címzésmódot tartalmazza a **cím közvetlen címzése** kivételével.

A **tároló** kategóriába tartozik mind az a címzésmód, amelyek nem közvetlenül egy regiszterre vonatkoznak.

Az **ellenőrző** kategóriába sorolhatók mindazok a címzésmódok, amelyek a tárolóra vonatkoznak, nincsenek elő- vagy utóinkrementálási tulajdonságaik, és nem közvetlen konstans adatokból állnak.

A **változtatható** kategóriába soroljuk mindazokat a címzésmódokat, amelyek nem közvetlenek (konstans), és amelyek az adatok tartományában vannak. Ez tehát a PC-relatív csoport valamennyi címzésmódját kizárja.

Ezeket összefoglaltuk egy táblázatban:

Címzésmód	Módus	Reg.	ADATOK	TÁROLÓ	ELLENŐRZŐ	VÁLTOZTATHATÓ	Szintaxis	Csak a 68020-as
Adatregiszter közvetlen	000	Nr.	X	–	–	–	Dn	
Címregiszter közvetlen	001	Nr.	–	–	–	X	An	
Címregiszter közvetett	010	Nr.	X	X	X	X	(An)	
Címregiszter közvetett, utólagos inkrementálással	011	Nr.	X	X	–	X	(An)+	
Címregiszter közvetett, előzetes dekrementálással	100	Nr.	X	X	–	X	–(An)	
Címregiszter közvetett, címkülönbséggel	101	Nr.	X	X	X	X	d_{16}(An)	
Címregiszter közvetett, 8 bites címkülönbséggel	110	Nr.	X	X	X	X	d_8(An, Rn)	
Címregiszter közvetett, indexszel	110	Nr.	X	X	X	X	bd(An, Rn)	*
Tároló közvetett, utólagos indexszel	110	Nr.	X	X	X	X	([bd, An] Rn, od)	*
Tároló közvetett, előzetes indexszel	110	Nr.	X	X	X	X	([bd, An, Rn]od)	*
Abszolút, rövid	111	000	X	X	X	X	xxxx. V	
Abszolút, hosszú	111	001	X	X	X	X	xxxxxxxx. L	
Programszámláló közvetett, 16 bites címkülönbséggel	111	101	X	X	X	–	d_{16}(PC)	
Programszámláló közvetett, 8 bites indexszel	111	011	X	X	X	–	d_8(PC, Rn)	
Programszámláló közvetett, indexszel	111	011	X	X	X	–	bd(PC, Rn)	*
PC-tároló közvetett, utólagos indexszel	111	011	X	X	X	–	([bd, PC]Rn, od)	*
PC-tároló közvetett, előzetes indexszel	111	011	X	X	X	–	([bd, PC, Rn]od)	*
Közvetlen	111	100	X	X	–	–	„adatok”	

Most már ahhoz, hogy minden egyes utasításhoz ki tudjuk választani az arra érvényes címzési módokat, csupán csak az érvényes címzésmódok kategóriáját vagy kategóriáit kell magadni. Két kategória kombinációja is elképzelhető, mint pl. az **adatok*változtatható,** ahol a * jel egy logikai ÉS-kapcsolatot jelent. Ilyen esetben tehát mind az a címzésmód érvényes, amely benne van mind az **adatok,** mind a **változtatható** csoportban. Ezáltal elmaradtak a programszámlálóhoz képest relatív hozzáférésfajták csakúgy, mint a közvetlen adatok, vagy a címzésregiszterhez való közvetlen hozzáférés.

5.2 Kapcsolók (Flags)

Kapcsolók alatt (flagek, zászlók) a CPU azon állapotjeleit értjük, amelyeket a processzor fejlesztői a felhasználó számára hozzáférhetővé tett. Ezeket általában ugyanúgy használjuk a korrekt matematikai feldolgozásokhoz, mint a program ellenőrzéséhez, tehát feltételes ugrásokhoz, alprogramok hívásához és hasonlókhoz.

A 68000-es CPU-nál öt kapcsoló áll a felhasználó rendelkezésére:

X	Ennek a bővítő (extension) bitnek az állapotát általában matematikai műveletek, mint összeadás, kivonás, eltolási és elforgatási műveletek befolyásolják. A matematikai műveleteknél a fő alkalmazási területe tulajdonképpen az, hogy négyes-szók is feldolgozhatók legyenek anélkül, hogy az ennek során esetleg igénybe vett, és így megváltozott állapotú átvitelbitet (Carry) tárolni kellene.
N	Ennek a Negatív bitnek az értéke normál esetben akkor 1, ha az operandus legnagyobb helyiértékű bitje 1, azaz az operandus kettes komplemensben negatív számnak tekintendő.
Z	A Zero bit normál esetben azt mutatja meg, hogy az eredmény (vagy vizsgáló műveleknél a forrás) nulla volt. Ha ez így volt, akkor a Z kapcsoló értéke 1.
V	A túlcsordulás (oVerflow) bit értéke normál körülmények között akkor 1, ha a legutolsó műveletnél operandustúlcsordulás fordult elő.
C	Az átvitel bit (Carry) az X bithez hasonlóan állítódik, azzal ellentétben azonban minden egyes matematikai-logikai művelet során változik.

Ebből az 5 bitből egy matematikai művelet, mint pl. kivonás, szorzás vagy összehasonlítás után a művelet körülbelüli kimenetele könnyedén megállapítható. Ezeken kívül, a program vezérlésére, tehát feltételes ugrásokhoz, alprogramok hívásához lekérdezési lehetőségként 16 különböző kód áll rendelkezésre:

A GE, LT, GT és LE feltételek valamennyiét a kettes komplemensben történő aritmetikai műveletekhez kell számítani, a két túlcsordulási lekérdezés is adott esetben ide számítható.

Mnemonik	**Feltétel**	**Kód**	**Bitkombináció**
T	True (igaz)	0000	mindig
F	False (hamis)	0001	soha
HI	High (magas)	0010	/C*/Z
LS	Low or Same	0011	C+Z
CC/HS	Carry Clear (Higher or Same) (átvitelt törölni/nagyobb vagy ua.)	0100	/C
CS/LO	Carry Set/Lower (átvitelt bekapcsolni/alacsonyabb)	0101	C
NE	Not Equal (nem egyenlő)	0110	/Z
EQ	EQual (egyenlő)	0111	Z
VC	Overflow Clear (túlcsordulást törölni)	1000	/V
VS	Overflow Set (túlcsordulást beállítani)	1001	V
PL	PLus	1010	/N
MI	MInus	1011	N
GE	Greater or Equal (nagyobb vagy egyenlő)	1100	N*V+/N*/V
LT	Less Than (kisebb mint)	1101	N*/V+/N*V
GT	Greater Than (nagyobb mint)	1110	N*V*/Z+/N*/V*/Z
LE	Less or Equal (kisebb vagy egyenlő)	1111	Z+N*/V+/N*V

* = logikai ÉS + = logikai VAGY
/ = logikai NEM

5.3 Ciklusmód (Loop)

A 68010-es és a 68020-as CPU-nak ezt az üzemmódját nem egy explicit utasítás váltja ki, hanem a már a 68000-esben meglevő DB_{CC} utasításra vonatkozik. Ez az utasítás először megvizsgálja a megadott feltételt. Ha ez teljesült, akkor a programvégrehajtás folytatódik a következő utasítással. Ellenkező esetben dekrementálja a megadott szót. Ha a szó értéke elérte a −1-et, akkor rátér a következő utasításra, egyébként elvégzi a magadott ugrást.

Példa:

```
           MOVE.W   hossz, D0
           MOVE.L   # kezdet, A0
           MOVE.L   # cél, A1
transp:    MOVE.W   A0 +, A1 +
           DBEQ     D0, transp
```

Ez a ciklus byte-okat (hossz) visz a (kezdet)-ről a (cél)-ba, és behatárolja, hogy vagy a hossz = −1, vagy pedig a legutolsó átvitt byte nulla volt.

Bár ilyen ciklust már a 68000-essel is el lehetett végeztetni, ez a 68010-esnél a prefetch-mechanizmusnak köszönhetően különösen hatékonyan alkalmazható. A prefetch-mechanizmus ennél és a későbbi processzoroknál lehetővé teszi a ciklusnak a processzoron belüli feldolgozást, és nem kell a mindenkori műveleti kódokat újra és újra betölteni. Ahhoz, hogy a prefetch e tulajdonságait ki lehessen használni, a ciklus (loop) utasításnak meg kell felelnie a következő feltételeknek:

Egyrészt az ugrástávolság maximum −4 lehet. Ez azt jelenti, hogy az utasításra a DB_{CC} előtt rá kell ugrani. Másrészt a DB_{CC} előtti utasításnak egy, a ciklust végrehajtani képes utasításnak kell lennie. Ezek a következők:

Utasítás	**Lehetséges címzésmódok**	
MOVE.X	(Ay),(Ax)+	−(Ay),(Ax)
	(Ay),(Ax)+	−(Ay),(Ax)+
	(Ay),−(Ax)	−(Ay),−(Ax)
	(Ay)+,−(Ax)	Ry,(Ax)
	(Ay)+,(Ax)+	Ry,(Ax)+
	(Ay)+,−(Ax)	
ADD.X	(Ay),DX	Dx,(Ay)
AND.X	(Ay)+,Dx	Dx,(Ay)+
CMP.X	−(Ay),Dx	Dx,−(Ay)
OR.X		
SUB.X		

Utasítás	**Lehetséges címzésmódok**
ADDA.X	(Ay),Ax
CMPA.X	−(Ay),Ax
SUBA.X	(Ay)+,Ax
ABCD	−(Ay),−(Ax)
ADDX.X	
SBCD	
SUBX.X	
CMP.X	(Ay)+,(Ax)+
CLR.X	(Ay)
NEG.X	(Ay)+
NEGX.X	−(Ay)
NOT.X	
TST.X	
NBCD	
ASL.W	(Ay), eggyel
ASR.W	(Ay)+, eggyel
LSL.W	−(Ay), eggyel
LSR.W	
ROL.W	
RDR.W	
ROXL.W	
ROXR.W	

Természetesen a 68000-es és a 68008-as processzorokkal is felépíthetők ilyen ciklusok. De csak a 68010-es tartja a belső regisztereiben az utasításszavakat, és nem kell ezeket minden ciklusfordulóban újra betölteni. A 68000-es és a 68008-as processzoroknál az RTE utasítás csak buszhibakizárás után nem engedélyezi a ciklus korrekt folytatását.

6. AZ UTASÍTÁSKÉSZLET

(Az egyes utasítások ábécé sorrendben követik egymást, és a 68000-es, 68008-as, 68010-es, 68012-es és a 68020-as processzorokra vonatkoznak.)

A B C D

$\text{cél}_{10} + \text{forrás}_{10} + X \rightarrow \text{cél}$

Binárisan kódolt decimális számok összeadása bővítő bittel
Add Binary Coded Decimals With Extend

Művelet	**Assembler szintaxis**
ABCD Dx-et a Dy-hoz	ABCD Dx, Dy
ABCD -(ind.)Ax-et -(ind.)Ay-hoz	ABCD –(Ax), –(Ay)
Operandus hossza:	csak byte

Kapcsolók:

X	N	Z	V	C
*	U	*	U	*

X értéke 1 lesz, ha decimális átvitel keletkezik, egyébként törlődik.
N nem határozható meg.
Z értéke 1 lesz, ha az eredmény nulla, egyébként nem változik.
V nem határozható meg.
C értéke 1 lesz, ha decimális átvitel keletkezik, egyébként törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	0	0	reg. y			1	0	0	0	0	T	reg. x		

reg. y	annak a cím-, ill. adatregiszternek a számát tartalmazza, amely célként van megadva.
reg. x	annak a cím-, ill. adatregiszternek a számát tartalmazza, amely forrásként van megadva.
T	specifikálja a cél- és a forráscímeket. Ha T = 0, akkor a Reg. x és a Reg. y adatregiszterekként, egyébként címregiszterekként kerülnek értelmezésre.

Az utasítás leírása:

Összeadásra kerül a forrás és a cél tartalma, az eredmény a célban kerül elhelyezésre, a kapcsolók értékei változnak. A következő két címzésmód áll rendelkezésre:

Adatregiszter az adatregiszterhez: az utasításszóban megadott regiszterek tartalmazzák a forrás- és a célinformációt.

Közvetett, előzetes dekrementálással címregisztereken keresztül: az operandusokat a processzor a tár azon helyeiről veszi elő, amelyekre az utasításszóban megadott címregiszterek 1-gyel való csökkentés után mutatnak.

Bináris összeadás
Add binary

Művelet	**Assembler szintaxis**
ADD Dn-t "ea"-hoz	ADD.X Dn, "ea"
ADD "ea"-t Dn-hez	ADD.X "ea", Dn
Operandus hossza:	byte, szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
*	*	*	*	*

X értéke 1 lesz, ha átvitel keletkezik, egyébként törlődik.
N értéke 1 lesz, ha az eredmény negatív, egyébként törlődik.
Z értéke 1 lesz, ha az eredmény zérus, egyébként törlődik.
V értéke 1 lesz, ha túlcsordulás lép fel, egyébként törlődik.
C értéke 1 lesz, ha átvitel keletkezik, egyébként törlődik.

Az utasítás formátuma:

bit	15 14 13 12	11 10 9	8 7 6	5 4 3 2 1 0
	1 1 0 1	reg.	módus	effektív cím

reg.	a nyolc adatregiszter egyikének a számát tartalmazza.
módus	meghatározza, hogy az adatregiszter cél vagy forrás, és az operandusok hosszát

Byte	**Szó**	**Kettős-szó**	**Irány**
000	001	010	Dn a cél
100	101	110	Dn a forrás

effektív cím: megadja a második operandus számára a címzésmódot, de az erre vonatkozó értékek a módustól függenek:

Ha "ea" a forrás, akkor egy kivétellel valamennyi címzésmód megengedett: ha az operandus hossza byte, akkor az An címzésmód (címregiszter közvetlen) nem megengedett.

Ha "ea" a cél, akkor a **tároló * változtatható** kategóriák valamennyi címzésmódja érvényes.

Az utasítás leírása:

Összeadásra kerül a forrás és a cél tartalma, az eredmény a célban kerül elhelyezésre. A kapcsolók értékei az eredménynek megfelelően változnak.

ADDA

cél + forrás –: cél

Bináris hozzáadás a címregiszterhez
ADD binary addressregister

Művelet	**Assembler szintaxis**
ADDA "ea" az An-hez	ADDA,X "ea", Dn
Operandus hossza:	szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	0	1	reg.			módus			effektív cím					

reg.	a nyolc címregiszter közül a mindenkori célregiszter számát tartalmazza.
módus	meghatározza az operandus hosszát: 001 = szó hosszúságú operandus. Azért, hogy a címet a helyes módon lehessen kiszámítani, a forrásoperandus előjelhelyesen 32 bitre bővül. 111 = kettős-szó hosszúságú operandus.

effektív cím: a forrásoperandust jelöli; valamennyi címzési mód érvényes.

Az utasítás leírása:

Az ADDA utasítás az ADD utasítástól szerkezetét tekintve csak a módusban különbözik. Az ADD utasításnál az x00, x01 és x10 értékek kerülnek felhasználásra. Így az ADDA utasítást – nemcsak szemantikailag – különleges címzési módú ADD utasításként tekinthetjük.

Közvetlen érték hozzáadása
ADD Immediate

Művelet	**Assembler szintaxis**
ADDI "data" az "ea"-hoz	ADDI.X # "data","ea"
Operandus hossza:	byte, szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
*	*	*	*	*

X értéke 1 lesz, ha átvitel keletkezik, egyébként törlődik.
N értéke 1 lesz, ha az eredmény negatív, egyébként törlődik.
Z értéke 1 lesz, ha az eredmény zérus, egyébként törlődik.
V értéke 1 lesz, ha túlcsordulás következik be, egyébként törlődik.
C értéke 1 lesz, ha átvitel keletkezik, egyébként törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	1	1	0	hossz		effektív cím					

Az utasításszóra egy szó következik, ha az operandus hossza byte vagy szó, és kettős-szó következik, ha az operandus kettős-szó hosszúságú. Ha az operandus hossza byte, akkor a szónak csak a legalsó nyolc bitje (0...7 bit) szignifikáns.

hossz	három értéket vehet fel: 00 = byte-művelet 01 = szóművelet 10 = kettős-szó művelet
effektív cím	a céloperandust határozza meg; csak az **adatok * változtatható** címzésmódok használhatók.

Az utasítás leírása:

Ez az utasítás egy állandó értéket ad hozzá binárisan egy megadandó célon levő értékhez, és az illető kapcsolók értékét megfelelően módosítja. Az állandó érték a tárban közvetlenül az utasításszó után áll, és lehet byte, szó és kettősszó hosszúságú. A byte hosszúságú operandusok a tárban szó hosszúságú címeket foglalnak el.

A D D Q cél+adatok –: cél

3 bit szélességű érték közvetlen, gyors hozzáadása
ADD immediate Quick

Művelet	**Assembler szintaxis**
ADDQ "data" az "ea"-hoz	ADDQ.X # "data","ea"
Operandus hossza:	byte, szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
*	*	*	*	*

X értéke 1 lesz, ha egy átvitel keletkezik, egyébként törlődik.
N értéke 1 lesz, ha az eredmény negatív, egyébként törlődik.
Z értéke 1 lesz, ha az eredmény zérus, egyébként törlődik.
V értéke 1 lesz, ha túlcsordulás következik be, egyébként törlődik.
C értéke 1 lesz, ha átvitel keletkezik, egyébként törlődik.

Az utasítás formátuma:

bit	15 14 13 12	11 10	9 8	7 6	5 4 3 2 1 0
	0 1 0 1	adatok	0	hossz	effektív cím

adatok	ez az a 3 bit szélességű érték, amit az "ea"-hoz hozzá kell adni.
hossz	három értéket vehet fel: 00 = byte-művelet 01 = szóművelet 10 = kettős-szó művelet
effektív cím	a céloperandust határozza meg; itt csak a **változtatható** kategóriájú címzésmódok használhatók. Byte-művelet esetén még az An címzési mód (címregiszter közvetlen) is tiltott.

Az utasítás leírása:

Ez az utasítás egy állandó értéket ad hozzá binárisan egy megadandó célon levő értékhez, és az illető kapcsolók értékét megfelelően módosítja. Az állandó érték (amelynek értéke 0...7 között lehet – maximum 3 bit! –, közvetlenül az utasításszóban van, és byte, szó vagy kettős-szó hosszúságú értékekhez adható hozzá.

Bináris összeadás a bővítő bittel
ADD binary with eXtend

Művelet	**Assembler szintaxis**
ADDX Dx-et a Dy-hoz	ADDX.X Dx, Dy
ADDX -(ind.)Ax-et az -(ind.)Ay-hoz	ADDX.X –(Ax), –(Ay)
Operandus hossza:	byte, szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
*	*	*	*	*

X értéke 1 lesz, ha átvitel keletkezik, egyébként törlődik.
N értéke 1 lesz, ha az eredmény negatív, egyébként törlődik.
Z értéke 1 lesz, ha az eredmény zérus, egyébként törlődik.
V értéke 1 lesz, ha túlcsordulás lép fel, egyébként törlődik.
C értéke 1 lesz, ha átvitel keletkezik, egyébként törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11 10 9	8	7 6	5	4	3	2 1 0
	1	1	0	1	reg. y	1	hossz	0	0	T	reg. x

reg.	annak a cím-, ill. adatregiszternek a számát tartalmazza, amely célként van megadva.
hossz	három értéket vehet fel: 00 = byte-művelet 01 = szóművelet 10 = kettős-szó művelet
reg. x	annak a cím-, ill. adatregiszternek a számát tartalmazza, amely forrásként van megadva.
T	A cél- és a forráscímet specifikálja. Ha T = 0, akkor a reg. x és reg. y adatregiszterek, ha T = 1, akkor címregiszterek értendők.

Az utasítás leírása:

Összeadásra kerül a cél és a forrás tartalma, és a bővítő kapcsoló értéke. Az eredmény a célban kerül elhelyezésre, a kapcsolók változnak. Kétféle címzésmód áll rendelkezésre:

Adatregiszter adatregiszterhez: az utasításszóban megadott regiszterek tartalmazzák a forrás- és a célinformációt.

Közvetett, előzetes dekrementálással a címregiszteren keresztül: az operandusok a tár azon helyeiről kerülnek elővételre, amelyekre az utasításszóban megadott címregiszterek 1-gyel való csökkentés után mutatnak.

logikai ÉS kapcsolat
logical AND

Művelet	**Assembler szintaxis**
AND Dn és "ea"	AND.X Dn, "ea"
AND "ea" és Dn	AND.X "ea", Dn

Operandus hossza: byte, szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
–	*	*	0	0

X nem változik
N értéke 1 lesz, ha a legmagasabb helyiértékű bit értéke 1, egyébként törlődik.
Z értéke 1 lesz, ha az eredmény zérus, egyébként törlődik.
V törlődik.
C törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	0	0	reg.			módus			effektív cím					

reg. a nyolc adatregiszter egyikének a számát tartalmazza.

módus meghatározza, hogy az adatregiszter cél vagy forrás-e, és az operandusok hosszát:

byte	**szó**	**kettős-szó**	**irány**
000	001	010	Dn a cél
100	101	110	Dn a forrás

effektív cím megadja a második operandusra a címzési módot, de az erre vonatkozó értékek a módustól függnek:

Ha "ea" a forrás, akkor csak az **adatok** címzésmód-kategória a megengedett.

Ha "ea" a cél, akkor a **tároló * változtatható** kategóriák valamennyi címzésmódja megengedett.

Az utasítás leírása:

A forrás és a cél tartalma között egy logikai ÉS művelet kerül elvégzésre, az eredmény a célban kerül elhelyezésre. A kapcsolók az eredménynek megfelelően változnak.

logikai ÉS kapcsolat közvetlen értékkel
logical AND with Immediate value

Művelet	**Assembler szintaxis**
ANDI "data" az "ea"-val	ANDI.X # "data", "ea"
Operandus hossza:	byte, szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
–	*	*	0	0

X nem változik
N értéke 1 lesz, ha az eredmény legnagyobb helyiértékű bitje 1, egyébként törlődik.
Z értéke 1 lesz, ha az eredmény zérus, egyébként törlődik.
V törlődik.
C törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	1	0	hossz		effektív cím					

Az utasításszót egy szó követi, ha az operandus hossza byte vagy szó, illetve kettős-szó követi, ha az operandus hossza kettős-szó. Ha az operandus hossza byte, akkor az utasításszót követő szónak csak a legalsó 8 bitje (0...7 bit) szignifikáns.

hossz	három értéket vehet fel: 00 = byte-művelet 01 = szóművelet 10 = kettős-szó művelet
effektív cím	a céloperandust határozza meg; az **adatok * változtatható** kategóriákba tartozó címzésmódok a megengedettek, az állapotregiszterrel kapcsolatban a **közvetlen** is.

Az utasítás leírása:

Ez az utasítás egy logikai ÉS műveletet végez el egy állandó érték és egy megadandó célon levő érték között, és beállítja a szükséges kapcsolókat. Az állandó érték a tárban közvetlenül az utasításszó után helyezkedik el, a hossza lehet byte, szó vagy kettős-szó. A byte hosszúságú operandusok szó hosszúságú címeket foglalnak el.

Ha a cél esetén az állapotregiszterről van szó, akkor ehhez kétféle módon lehet hozzáférni:

Ha byte-műveletről van szó, akkor az állapotregiszternek csak a USER-fele kerül felhasználásra.
Ha szóműveletről van szó, akkor ez az utasítás egy **privilegizált utasítás.**

egy regiszter bitenkénti, aritmetikai eltolása balra/jobbra
Arithmetic Shift Left/Right

Művelet	**Assembler szintaxis**
ASx Dx-et Dy hellyel	ASx.X Dy,Dx
ASx Dx-et "data" hellyel	ASx.X # "data", Dx
ASx "ea"-et 1 hellyel	ASx.X "ea"
Operandus hossza:	byte, szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
*	*	*	*	*

X azt az értéket veszi fel, ami az operandus legutoljára kitolt bitjének az értéke volt. Ha azon a helyen zérus érték volt, akkor az X nem változik.
N értéke 1 lesz, ha az eredmény legnagyobb helyiértékű bitje 1, egyébként törlődik.
Z értéke 1 lesz, ha az eredmény zérus, egyébként törlődik.
V értéke 1 lesz, ha az operandus legnagyobb helyiértékű bitjének értéke változik, egyébként törlődik.
C ugyanaz, mint X-nél.

Az utasítás formátuma egy adatregiszter eltolásához:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	0	reg.			dr	hossz		i	0	0	regiszter		

reg. a nyolc adatregiszter (Dy) közül annak a számát tartalmazza, amelyik az eltolások számát adja meg, ill. meghatározza az abszolút értéket (data) – (az i-bit értékétől függően!). Az abszolút érték a számát tekintve 0...7 között lehet (3 bit), és a 0 érték nyolc eltolásnak felel meg.

dr	Ez a bit az eltolások irányát (DiRection) adja meg: 0 = eltolás jobbra 1 = eltolás balra
hossz	három értéket vehet fel: 00 = byte-művelet 01 = szóművelet 10 = kettős-szó művelet
i	eldönti, hogy a reg. egy regisztert vagy egy abszolút (közvetlen) értéket jelent-e: 0 = közvetlen érték 1 = a reg. jelentése adatregiszter
regiszter	ez a három bit azt az adatregisztert jelöli ki, amelynek a tartalmát el kell tolni (a cél címe).

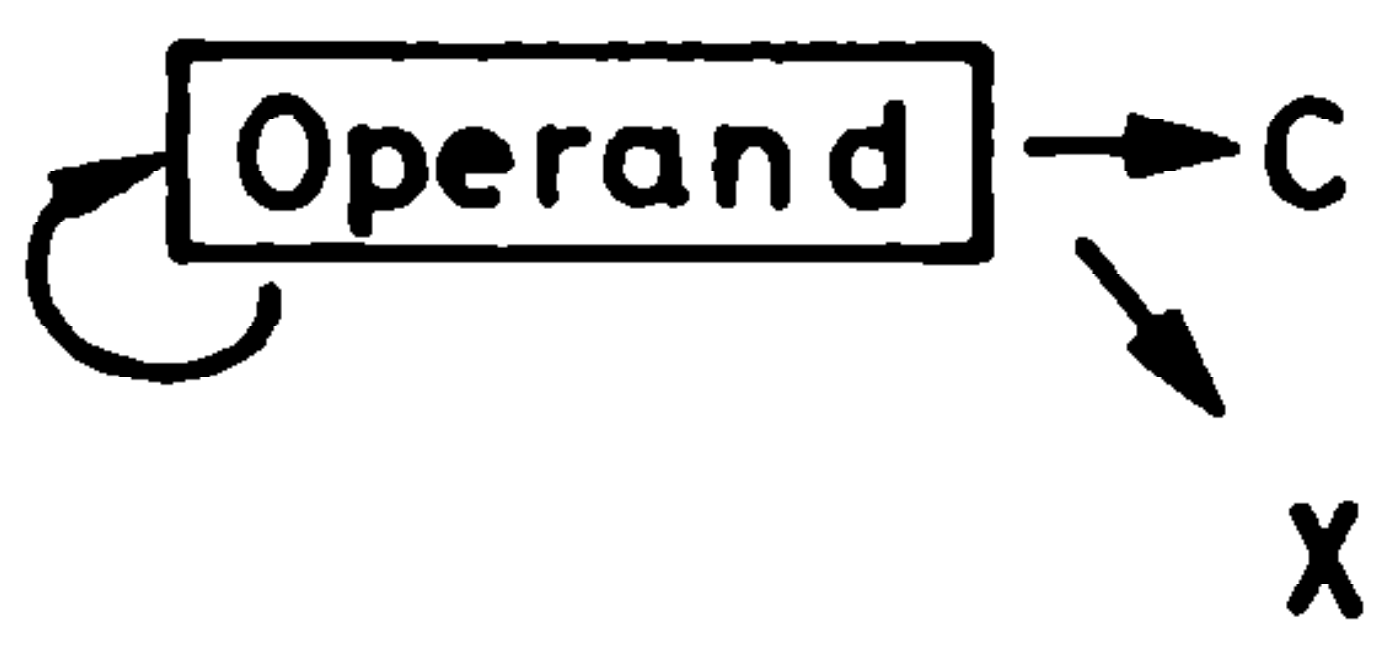
Az utasítás leírása:

Az utasítás hatására a megadandó cél tartalma az ugyancsak megadandó számú bináris hellyel jobbra, ill. balra tolódik el.

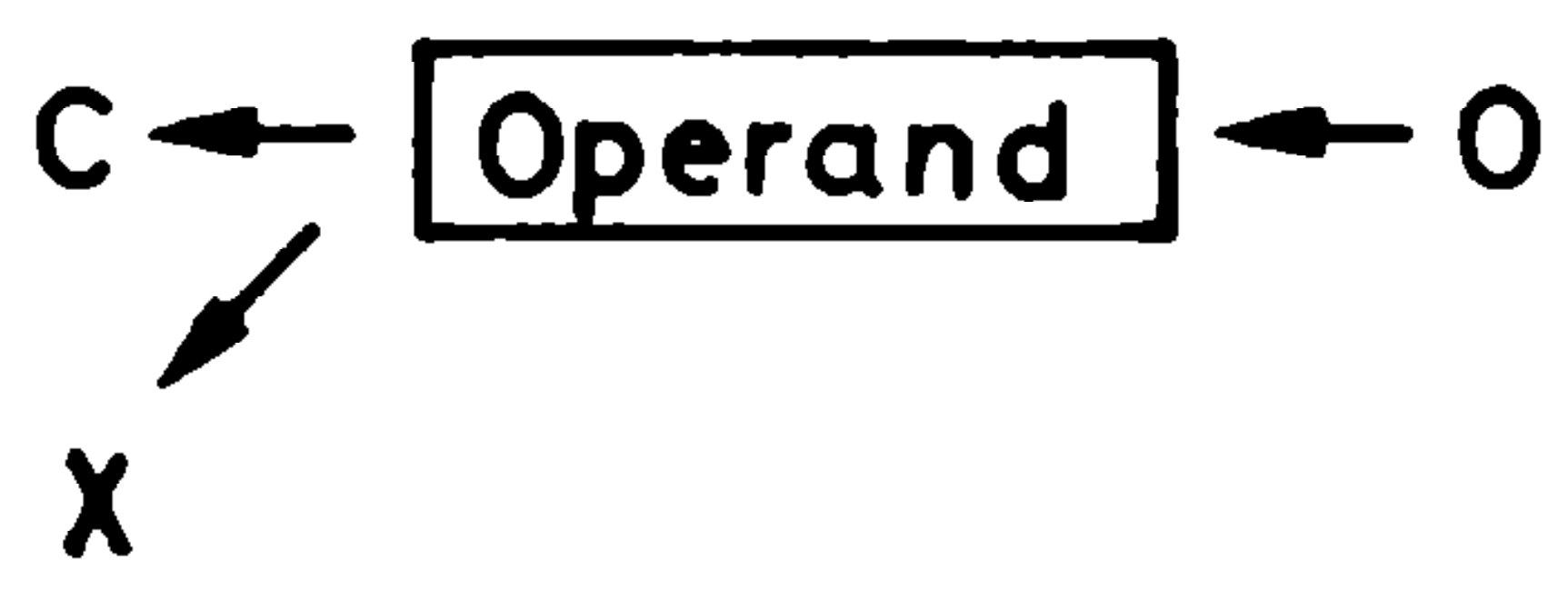
Az eltolás irányát az assembler nyelv különböző szintaxissal jelöli (ASL/ASR), a két esetben létrehozott kódok azonban csak egy bitben különböznek egymástól.
A cél címeként egy adatregiszter adható meg. Ebben az esetben a cél címén (azaz az adatregiszteren) kívül még azt is meg kell adni, hogy az eltolás hány hellyel történjen meg. Ezt vagy egy konstans formájában, vagy egy másik adatregiszterben lehet megadni.

Ha viszont célként nem specifikálunk adatregisztert, akkor célként a **tároló * változtatható** tartományból valamennyi címzésmód megengedett, de ekkor az eltolandó helyek száma nem adható meg. Ennek az értéke ebben az esetben mindig 1.

Ügyelni kell továbbá arra, hogy az operandus hosszát (byte, szó, kettős-szó) csak regiszter eltolása esetén lehet megválasztani. Ha a művelet a tárra vonatkozik, akkor az operandus mindig szó hosszúságú.



A S R



A S L

Az utasítás formátuma a tárban való eltoláshoz:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	0	0	0	0	dr	1	1	effektív cím					

dr	ez a bit az eltolás irányát (DiRection) adja meg: 0 = eltolás jobbra 1 = eltolás balra
effektív cím	a céloperandust határozza meg; az **adatok + változtatható** valamennyi címzésmódja lehetséges.

B_{cc} ha $_{cc}$ igaz, akkor PC + távolság –: PC

relatív ugrás, ha a feltétel(ek) teljesült(ek)
Branch if Condition-Codes true

Művelet	**Assembler szintaxis**
B_{cc} a "címké"-hez	B_{cc} "címke"
Operandus hossza:	byte vagy szó

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	1	0	feltétel				8 bites címkülönbség							

Az utasításszó után egy másik szó következik, ha a 8 bites címkülönbség nulla. Ebben az esetben egy „távoli" relatív ugrással van dolgunk, e távolság értékét ugyancsak ez a követő szó tartalmazza.
Feltétel: itt a következőkben leírt feltételkombinációk egyike áll.

Az utasítás leírása:

Az utasítás végrehajtásának megkezdésekor felülvizsgálatra kerül az állapotregiszter a megadott $_{cc}$ feltételre vonatkozóan. Ha a feltétel érvényes, tehát logikailag igaz, akkor a PC programszámláló a megadott címkülönbség értékével megváltozik. Ha a címkülönbség a −128...+127 tartományban van (az ugrási utasítást követő első byte-tól számítva), akkor a különbség egy 8 bites értékben az utasításban implicite továbbításra kerül. Ha az ugrás távolsága −32 768...+32 767 között van, akkor ez egy „távoli ugrás"-nak számít, és a különbség szó méretben az utasításszó mögött kerül elhelyezésre. Ebben az esetben a 8 bites érték mindig nulla, és a cím kiszámítása a különbséget tartalmazó szó utáni címtől kezdődik.

A $_{cc}$ feltételként a következő 14 bitkombináció valamelyike adható meg:

Név	**Kód**	**Jelentése**	**Név**	**Kód**	**Jelentése**
CC	0100	Carry=0	LS	0011	nem nagyobb
CS	0101	Carry=1	LT	1101	kisebb
EQ	0111	egyenlő	MI	1011	negatív
GE	1100	nagyobb/egyenlő	NE	0110	nem egyenlő
GT	1110	nagyobb mint 0	PL	1010	pozitív
HI	0010	nagyobb	VC	1000	túlcsordulás=0
LE	1111	kisebb/egyenlő	VS	1001	túlcsordulás=1

E feltételek pontosabb leírását, különösen a kapcsolókkal való ábrázolásukat illetően utalunk e fejezet elején leírtakra.

B C H G

$!(\text{cél}_{\text{bitszám}}) - : \text{cél}_{\text{bitszám}} - : Z$

egy bit vizsgálata és a komplemensének visszaírása
test Bit and CHanGe

Művelet	**Assembler szintaxis**
BCHG Dx, "ea"	BCHG Dx, "ea"
BCHG "adat", "ea"	BCHG # "adat", "ea"
Operandus hossza:	byte vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
–	–	*	–	–

X nem változik.
N nem változik.
Z értéke 1 lesz, ha a vizsgálandó bit értéke zérus volt, egyébként törlődik.
V nem változik.
C nem változik.

Az utasítás formátuma, ha a bit száma a regiszterben van:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	Dx			1	0	1	effektív cím					

Dx: az az adatregiszter, amely az illető bit számát tartalmazza.

effektív cím: meghatározza a céloperandust; az **adatok * változtatható** címzésmódok mindegyike használható.

Az utasítás leírása:

Ennek az utasításnak a feldolgozásakor elsőként átmásolódik a zérókapcsolóba (Z-flag) annak a bitnek a komplemense, amelyet a Dx vagy egy közvetlen érték, és az effektív cím határoz meg. Ezután ez a komplemens visszaíródik arra a helyre, ahonnan a bit kiolvasása történt.

Ha a cél címe a nyolc adatregiszter valamelyikében van, akkor bitmaszkként a megadott bitszám modulo 32-es értékét használjuk, egyébként, tehát ha a tárban levő valamelyik bitet vizsgáljuk, akkor a megadott bitszám modulo 8-as értékét dolgozzuk fel. Erre a megkülönböztetésre azért van szükség, hogy egyrészt egy CPU-regiszterben egy komplett, 32 bit szélességű kettős-szó feldolgozható legyen, másrészt viszont azért, hogy különleges buszkoncepciók megvalósításához akár egyetlen egy, tárbeli byte-ot is célba lehessen venni. Ha a bitszám közvetlen értékként van megadva, akkor ez a műveleti kódot követő első szó legalsó 8 bitjén helyezkedik el.

Az utasítás formátuma, ha a bitszám közvetlen érték:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	1	0	0	0	0	1	effektív cím					

BCLR

$!(\text{cél}_{\text{bitszám}}) -: Z$

$0 -: \text{cél}_{\text{bitszám}}$

egy bit vizsgálata és törlése
Bit test and CLeaR

Művelet	**Assembler szintaxis**
BCLR D, "ea"	BCLR Dx, "ea"
BCLR "adat", "ea"	BCLR # "adat", "ea"
Operandus hossza:	byte vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
–	–	*	–	–

X nem változik.
N nem változik.
Z értéke 1 lesz, ha a vizsgálandó bit zérus volt, egyébként törlődik.
V nem változik.
C nem változik.

Az utasítás formátuma, ha a bit száma a regiszterben van:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	Dx			1	1	0	effektív cím					

Dx:	az az adatregiszter, amely az illető bit számát tartalmazza.
effektív cím:	meghatározza a céloperandust; az **adatok * változtatható** címzésmódok mindegyike használható.

Az utasítás leírása:

Ennek az utasításnak a feldolgozásakor elsőként átmásolódik a zérókapcsolóba (Z-flag) annak a bitnek a komplemense, amelyet a Dx vagy egy közvetlen érték, és az effektív cím határoz meg. Ezután egy nulla íródik arra a helyre, ahonnan az előző olvasás történt.

Ha a cél címe a nyolc adatregiszter valamelyikében van, akkor bitmaszkként a megadott bitszám modulo 32-es értékét használjuk, egyébként, tehát ha a tárban levő valamelyik bitet vizsgáljuk, akkor a megadott bitszám modulo 8-as értékét dolgozzuk fel. Erre a megkülönböztetésre azért van szükség, hogy egyrészt egy CPU-regiszterben egy komplett, 32 bit szélességű kettős-szó feldolgozható legyen, másrészt viszont azért, hogy különleges buszkoncepciók megvalósításához akár egyetlen egy, tárbeli byte-ot is célba lehessen venni. Ha a bitszám közvetlen értékként van megadva, akkor ez a műveleti kódot követő első szó legalsó 8 bitjén helyezkedik el. Feltétlenül vegye figyelembe a BTST utasítás leírását!

Az utasítás formátuma, ha a bitszám közvetlen érték:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	1	0	0	0	1	0	effektív cím					

BFCHG $!(cél_{bitmező}$ –: $cél_{bitmező}$

bitmező–operandus vizsgálata és invertálása
Bit Field test and CHanGe
CSAK A 68020-AS PROCESSZOROKNÁL

Művelet	**Assembler szintaxis**
BFCHG "ea"	BFCHG offset: hossz
Operandus hossza:	bitmező

Kapcsolók:

X	N	Z	V	C
–	*	*	0	0

X nem változik.
N értéke 1 lesz, ha a vizsgálandó bitmező legmagasabb helyiértékű bitje magas, egyébként törlődik.
Z értéke 1 lesz, ha a vizsgálandó bitmező zérus volt, egyébként törlődik.
V törlődik.
C törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	0	1	0	1	0	1	1	effektív cím					

effektív cím	ez adja meg azt a célként szolgáló kettős-szót, amelyben a vizsgálandó bitmező van. A **vezérlés * változtatható** kategóriák valamennyi címzésmódja megengedett, valamint a Dx címzésmód, tehát az *adatregiszter közvetlen* is.
Do	0 = OFFSET, az abszolút offszetet tartalmazza. 1 = OFFSET, a 8...6. bitekben annak az adatregiszternek a regiszterszámát tartalmazza, amely regiszter a tulajdonképpeni offszetet tartalmazza.

offset Itt vagy egy 0 és 31 közé eső közvetlen érték, vagy egy 0 és 7 közé eső regiszterszám áll, azt az adatregisztert specifikálva, amely aztán egy bitmezőoffszetet ad meg a $-2^{31} \ldots 2^{31}-1$ tartományban. A két eset közüli választást a Do bit végzi el.

Az utasításszót egy újabb szó követi, amelyik az effektív címen belül a bitmezőt pontosabban meghatározza:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	Do	offset					Dw	hossz				

Dw 0 = a szónak ez a hossz-mezeje egy közvetlenül megadott hosszadatot tartalmaz.
1 = a szónak ez a hossz-mezeje annak az adatregiszternek a számát tartalmazza, amelyik a tulajdonképpeni távolsági adatot tartalmazza. Mivel ez az adat a regiszterben a $-2^{31} \ldots 2^{31}-1$ tartományt foghatja át, ennek a regisztertartalomnak csak a modulo 32-es értéke kerül effektív hosszként felhasználásra.

hossz ez a mező vagy egy 0...31 közé eső állandót, vagy annak az adatregiszternek a számát tartalmazza, amely a hosszra vonatkozó adatot tartalmazza. A két eset közüli választást a Dw bit végzi el.

Az utasítás leírása:

E művelet végrehajtásakor először felülvizsgálatra kerül a megadott bitmező a nullával való egyezőségre és a negativitásra vonatkozóan, hogy a kapcsolókat ennek megfelelően lehessen állítani. Ezután a megadott mező minden egyes bitje invertálásra kerül.

Mind a bitmező kezdete, mind annak hossza megadható adatregiszterekben vagy közvetlenül is.

Ha a hosszoperandus modulo 32-t nullának adjuk meg, akkor ez 32 bit hosszúságnak felel meg.

Ez a művelet csak a 68020-as processzorba van beépítve.

B F C L R 0 –: $\text{cél}_{\text{bitmező}}$

bitmező operandus vizsgálata és törlése
Bit Field test and CLear
CSAK A 68020-AS PROCESSZORNÁL

Művelet	**Assembler szintaxis**
BFCLR "ea"	BFCLR offset: hossz
Operandus hossza:	bitmező

Kapcsolók:

X	N	Z	V	C
–	*	*	–	–

X nem változik.
N értéke 1 lesz, ha a vizsgálandó bitmező legmagasabb helyiértékű bitje magas volt, egyébként törlődik.
Z értéke 1 lesz, ha a vizsgálandó bitmező zérus volt, egyébként törlődik.
V törlődik.
C törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	0	1	1	0	0	1	1	effektív cím					

effektív cím	ez adja meg azt a célként szolgáló kettős-szót, amelyben a vizsgálandó bitmező van. A **vezérlés * változtatható** kategóriák valamennyi címzésmódja megengedett, valamint a Dx címzésmód, tehát az *Adatregiszter közvetlen* is.
Do	0 = OFFSET, az abszolút offszetet tartalmazza. 1 = OFFSET, a 8...6. bitekben annak az adatregiszternek a regiszterszámát tartalmazza, amely regiszter a tulajdonképpeni offszetet tartalmazza.

offset	Itt vagy egy 0 és 31 közé eső közvetlen érték, vagy 0 és 7 közé eső regiszterszám áll, azt az adatregisztert specifikálva, amely aztán egy bitmezőoffszetet ad meg a $-2^{31}\ldots2^{31}-1$ tartományban. A két eset közüli választást a Do bit végzi el.

Az utasításszót egy újabb szó követi, amelyik az effektív címen belül a bitmezőt pontosabban meghatározza:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	Do	offset					Dw	hossz				

Dw	0 = a szónak a **hossz**-mezeje egy közvetlenül megadott hosszadatot tartalmaz. 1 = a szónak ez a **hossz**-mezeje annak az adatregiszternek a számát tartalmazza, amelyik a tulajdonképpeni távolsági adatot tartalmazza. Mivel ez az adat a regiszterben a $-2^{31}\ldots2^{31}-1$ tartományt foghatja át, ennek a regisztertartalomnak csak a modulo 32-es értéke kerül effektív hosszként felhasználásra.
hossz	ez a mező vagy egy 0...31 közé eső állandót, vagy vannak az adatregiszternek a számát tartalmazza, amely a hosszra vonatkozó adatot tartalmazza. A két eset közüli választást a Dw bit végzi el.

Az utasítás leírása:

E művelet végrehajtásakor először felülvizsgálatra kerül a megadott bitmező a nullával való egyezőségre és a negativitásra vonatkozóan, hogy a kapcsolókat ennek megfelelően lehessen állítani. Ezután a megadott mező minden egyes bitje törlésre kerül.

Mind a bitmező kezdete, mind annak hossza megadható adatregiszterekben vagy közvetlenül is.

Ha a hosszoperandus modulo 32-t nullának adjuk meg, akkor ez 32 bit hosszúságnak felel meg.

Ez a művelet csak a 68020-as processzorba van beépítve.

bitmező előjelhelyes betöltése adatregiszterbe
Bit Field EXTract Signed
CSAK A 68020-AS PROCESSZORNÁL

Művelet	**Assembler szintaxis**
BFEXTS "ea" Dx-be	BFEXTS offset: hossz, Dx
Operandus hossza:	bitmező

Kapcsolók:

X	N	Z	V	C
–	*	*	0	0

X nem változik.
N értéke 1 lesz, ha az eltávolítandó bitmező legmagasabb helyiértékű bitje magasra van állítva, egyébként törlődik.
Z értéke 1 lesz, ha az eltávolítandó bitmező zérus volt, egyébként törlődik.
V törlődik.
C törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	0	1	0	1	1	1	1	effektív cím					

Reg. Dx	itt annak az adatregiszternek a száma áll, amelybe az eltávolítandó bitmezőt tölteni kell.
effektív cím:	megadja azt a célként szolgáló kettős-szót, amelyben a vizsgálandó bitmező van. A **vezérlés** kategóriák valamennyi címzésmódja megengedett, valamint a Dx címzésmód, azaz az *adatregiszter közvetlen* is.
Do	0 = OFFSET, az abszolút offszetet tartalmazza. 1 = OFFSET, a 8...6 bitekben annak az adatregiszternek a számát tartalmazza, amelyik a tulajdonképpeni offszetet tartalmazza.

Az utasításszót egy újabb szó követi, amelyik az effektív címen belül a bitmezőt pontosabban meghatározza:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0 reg. Dx				Do	offset					Dw	hossz				

offset	Itt vagy 0 és 31 közé eső közvetlen érték, vagy 0 és 7 közé eső regiszterszám áll, azt az adatregisztert specifikálva, amely aztán egy bitmezőoffszetet ad meg a $-2^{31}...2^{31}-1$ tartományban. A két eset közüli választást a Do bit végzi el.
Dw	0 = a szónak a hosszmezeje egy közvetlenül megadott hosszadatot tartalmaz. 1 = a szónak ez a hosszmezeje annak az adatregiszternek a számát tartalmazza, amelyik a tulajdonképpeni távolsági adatot tartalmazza. Mivel ez az adat a regiszterben a $-2^{31}...2^{31}-1$ tartományt foghatja át, ennek a regisztertartalomnak csak a modulo 32-es értéke kerül effektív hosszként felhasználásra.
hossz	ez a mező vagy egy 0...31 közé eső állandót, vagy annak az adatregiszternek a számát tartalmazza, amely a hosszra vonatkozó adatot tartalmazza. A két eset közüli választást a Dw bit végzi el.

Az utasítás leírása:

E művelet végrehajtásakor először felülvizsgálatra kerül a megadott bitmező a nullával való egyezőségre és a negativitásra vonatkozóan, hogy a kapcsolókat ennek megfelelően lehessen állítani. Ezt követően a bitmező előjelhelyesen bővítésre, és a megadott céladatregiszterben elhelyezésre kerül. A bitmező legalacsonyabb helyiértékű bitje az adatregiszter 0. bitje utáni helyre kerül.

Mind a bitmező kezdete, mind annak hossza megadható adatregiszterekben, vagy közvetlenül is.

Ha a hosszoperandus modulo 32-t nullának adjuk meg, akkor ez 32 bit hosszúságnak felel meg.

Ez a művelet csak a 68020-as processzorba van beépítve.

bitmező betöltése adatmezőbe
Bit Field EXTract Unsigned
CSAK A 68020-AS PROCESSZORNÁL

Művelet	**Assembler szintaxis**
BFEXTU "ea" Dx-be	BFEXTU offset: hossz, Dx
Operandus hossza:	bitmező

Kapcsolók:

X	N	Z	V	C
–	*	*	0	0

X nem változik.
N értéke 1 lesz, ha az eltávolítandó bitmező legmagasabb helyiértékű bitje magasra volt állítva, egyébként törlődik.
Z értéke 1 lesz, ha az eltávolítandó bitmező zérus volt, egyébként törlődik.
V törlődik.
C törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	0	1	0	0	1	1	1	effektív cím					

reg. Dx	itt annak az adatregiszternek a száma áll, amelybe az eltávolítandó bitmezőt tölteni kell.
effektív cím:	megadja azt a célként szolgáló kettős-szót, amelyben a vizsgálandó bitmező van. A **vezérlés** kategóriák valamennyi címzésmódja megengedett, valamint a Dx címzésmód, azaz az *adatregiszter közvetlen* is.
Do	0 = OFFSET, az abszolút offszetet tartalmazza.

1 = OFFSET, a 8...6 bitekben annak az adatregiszternek a számát tartalmazza, amelyik a tulajdonképpeni offszetet tartalmazza.

Az utasításszót egy újabb szó követi, amelyik az effektív címen belül a bitmezőt pontosabban meghatározza:

bit	15 14 13 12	11	10 9 8 7 6	5	4 3 2 1 0
	0 reg. Dx	Do	offset	Dw	hossz

offset	Itt vagy 0 és 31 közé eső közvetlen érték, vagy 0 és 7 közé eső regiszterszám áll, azt az adatregisztert specifikálva, amely aztán egy bitmezőoffszetet ad meg a $-2^{31}...2^{31}-1$ tartományban. A két eset közüli választást a Do bit végzi el.
Dw	0 = a szónak a hosszmezeje egy közvetlenül megadott hosszadatot tartalmaz. 1 = a szónak ez a hosszmezeje annak az adatregiszternek a számát tartalmazza, amelyik a tulajdonképpeni távolsági adatot tartalmazza. Mivel ez az adat a regiszterben a $-2^{31}...2^{31}-1$ tartományt foghatja át, ennek a regisztertartalomnak csak a modulo 32-es értéke kerül effektív hosszként felhasználásra.
hossz	ez a mező vagy egy 0...31 közé eső állandót, vagy annak az adatregiszternek a számát tartalmazza, amely a hosszra vonatkozó adatot tartalmazza. A két eset közüli választást a Dw bit végzi el.

Az utasítás leírása:

E művelet végrehajtásakor először felülvizsgálatra kerül a megadott bitmező a nullával való egyezőségre és a negativitásra vonatkozóan, hogy a kapcsolókat ennek megfelelően lehessen állítani. Ezt követően a bitmező előjelének figyelmen kívül hagyásával a megadott céladatregiszterben elhelyezésre kerül.

A céladatregiszterben azok a bitek, amelyeket nem a bitmező „töltött fel", törlődnek.

Mind a bitmező kezdete, mind annak hossza megadható adatregiszterekben, vagy közvetlenül is.

Ha a hosszoperandus modulo 32-t nullának adjuk meg, akkor ez 32 bit hosszúságnak felel meg.

Ez a művelet csak a 68020-as processzorba van beépítve.

BFFFO (forrás$_{\text{bitmező}}$)szám –: Dx

a bitmezőben az első 1-es érték keresése, és helyzetének kijelzése.
Bit Field Find First One
CSAK A 68020-AS PROCESSZORNÁL

Művelet	**Assembler szintaxis**
BFFFO "ea" a Dx-ben	BFFFO offset: hossz, Dx
Operandus hossza:	bitmező

Kapcsolók:

X	N	Z	V	C
–	*	*	0	0

X nem változik.
N értéke 1 lesz, ha az átvizsgálandó bitmező legmagasabb helyiértékű bitje magasra volt állítva, egyébként törlődik.
Z értéke 1 lesz, ha az átvizsgálandó bitmező zérus volt, egyébként törlődik.
V törlődik.
C törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	0	1	1	0	1	1	1	effektív cím					

reg. Dx	itt annak az adatregiszternek a száma áll, amelybe a bit pozícióját be kell tölteni.
effektív cím:	megadja azt a célként szolgáló kettős-szót, amelyben a vizsgálandó bitmező van. A **vezérlés** kategóriák valamennyi címzésmódja megengedett, valamint a Dx címzésmód, azaz az *adatregiszter közvetlen* is.
Do	0 = OFFSET, az abszolút offszetet tartalmazza. 1 = OFFSET, a 8...6 bitekben annak az adatregiszternek a számát tartalmazza, amelyik a tulajdonképpeni offszetet tartalmazza.

Az utasításszót egy újabb szó követi, amelyik az effektív címen belül a bitmezőt pontosabban meghatározza:

bit	15 14 13 12	11	10 9 8 7 6	5	4 3 2 1 0
	0 reg. Dx	Do	offset	Dw	hossz

offset	Itt 0 és 31 közé eső közvetlen érték, vagy 0 és 7 közé eső regiszterszám áll, azt az adatregisztert specifikálva, amely aztán egy bitmezőoffszetet ad meg a $-2^{31}...2^{31}-1$ tartományban. A két eset közüli választást a Do bit végzi el.
Dw	0 = a szónak a hossz-mezeje egy közvetlenül megadott hosszadatot tartalmaz. 1 = a szónak ez a hossz-mezeje annak az adatregiszternek a számát tartalmazza, amelyik a tulajdonképpeni távolsági adatot tartalmazza. Mivel ez az adat a regiszterben a $-2^{31}...2^{31}-1$ tartományt foghatja át, ennek a regisztertartalomnak csak a modulo 32-es értéke kerül effektív hosszként felhasználásra.
hossz	ez a mező vagy 0...31 közé eső állandót, vagy annak az adatregiszternek a számát tartalmazza, amely a hosszra vonatkozó adatot tartalmazza. A két eset közüli választást a Dw bit végzi el.

Az utasítás leírása:

E művelet végrehajtásakor először felülvizsgálatra kerül a megadott bitmező a nullával való egyezőségre és a negativitásra vonatkozóan, hogy a kapcsolókat ennek megfelelően lehessen állítani. Ezt követően megkezdődik balról az első 1-es érték keresése. Ha sikerült egy 1-est találni, akkor ennek a pozíciója offszetként elhelyezésre kerül a céladatregiszterben. Ha a bitmező mindenütt nulla, akkor a visszatérési érték az offszet plusz az adatregiszterben levő távolság.

Mind a bitmező kezdete, mind annak hossza megadható adatregiszterekben, vagy közvetlenül is.

Ha a hosszoperandus modulo 32-t nullának adjuk meg, akkor az 32 bit hosszúságnak felel meg.

Ez a művelet csak a 68020-as processzorba van beépítve.

bitmező betöltése regiszterből, effektív cím szerint
Bit Field INSert
CSAK A 68020-AS PROCESSZORNÁL

Művelet	**Assembler szintaxis**
BFINS Dx az "ea" szerint Operandus hossza	BFINS Dx, offset: hossz bitmező

Kapcsolók

X	N	Z	V	C
–	*	*	0	0

X nem változik.
N értéke 1 lesz, ha a tárolandó bitmező legnagyobb helyiértékű bitje magasra volt állítva, egyébként törlődik.
Z értéke 1 lesz, ha a tárolandó bitmező zérus volt, egyébként törlődik.
V törlődik.
C törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	0	1	1	1	1	1	1	effektív cím					

reg. Dx	itt annak az adatregiszternek a száma áll, amelyben a tárolandó bitmező található.
effektív cím:	megadja azt a célként szolgáló kettős-szót, amelybe a bitmezőt el kell helyezni. A vezérlés + változtatható kategóriák valamennyi címzésmódja megengedett, továbbá a Dx címzésmód, azaz az *adatregiszter közvetlen* is.
Do	0 = OFFSET, az abszolút offszetet tartalmazza 1 = OFFSET, a 8...6 bitekben annak az adatregiszternek a számát tartalmazza, amelyik a tulajdonképpeni offszetet tartalmazza.

Az utasításszót egy újabb szó követi, amelyik az effektív címen belül a bitmezőt pontosabban meghatározza:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0 reg. Dx				Do	offset					Dw	hossz				

offset	Itt 0 és 31 közé eső közvetlen érték, vagy 0 és 7 közé eső regiszterszám áll, azt az adatregisztert specifikálva, amely aztán egy bitmezőoffszetet ad meg a $-2^{31}...2^{31}-1$ tartományban. A két eset közüli választást a Do bit végzi el.
Dw	0 = a szónak a hossz-mezeje egy közvetlenül megadott hosszadatot tartalmaz. 1 = a szónak ez a hossz-mezeje annak az adatregiszternek a számát tartalmazza, amelyik a tulajdonképpeni távolsági adatot tartalmazza. Mivel ez az adat a regiszterben a $-2^{31}...2^{31}-1$ tartományt foghatja át, ennek a regisztertartalomnak csak a modulo 32-es értéke kerül effektív hosszként felhasználásra.
hossz	ez a mező vagy egy 0...31 közé eső állandót, vagy annak az adatregiszternek a számát tartalmazza, amely a hosszra vonatkozó adatot tartalmazza. A két eset közüli választást a Dw bit végzi el.

Az utasítás leírása:

E művelet végrehajtásakor először felülvizsgálatra kerül a forrás-adatregiszterben megadott bitmező a nullával való egyezőségre és a negativitásra vonatkozóan, hogy a kapcsolókat ennek megfelelően lehessen állítani. Ezt követően a bitmező az effektív cím, az offszet és a hossz által megadott címen kerül elhelyezésre.

Mind a bitmező kezdete, mind annak hossza megadható adatregiszterekben, vagy közvetlenül is.

Ha a hosszoperandus modulo 32-t nullának adjuk meg, akkor az 32 bit hosszúságnak felel meg.

Ez a művelet csak a 68020-as processzorba van beépítve.

bitmező-operandus vizsgálata és magasra állítása
Bit Field SET
CSAK A 68020-AS PROCESSZORNÁL

Művelet	**Assembler szintaxis**
BFSET "ea" Operandus hossza:	BFSET offset: hossz bitmező

Kapcsolók:

X	N	Z	V	C
–	*	*	0	0

X nem változik.
N értéke 1 lesz, ha a magasra állítandó bitmező legmagasabb helyiértékű bitje magasra volt állítva, egyébként törlődik.
Z értéke 1 lesz, ha a magasra állítandó bitmező zérus volt, egyébként törlődik.
V törlődik.
C törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	0	1	1	1	0	1	1	effektív cím					

effektív cím:	megadja azt a célként szolgáló kettős-szót, amelyen a bitmezőt magasra kell állítani. A "**vezérlés * változtatható**" kategóriák valamennyi címzésmódja megengedett, valamint a Dx címzésmód, azaz az „Adatregiszter közvetlen" is.
Do	0 = OFFSET, az abszolút offszetet tartalmazza. 1 = OFFSET, a 8...6 bitekben annak az adatregiszternek a számát tartalmazza, amelyik a tulajdonképpeni offszetet tartalmazza.

offset	Itt 0 és 31 közé eső közvetlen érték, vagy 0 és 7 közé eső regiszterszám áll, azt az adatregisztert specifikálva, amely aztán egy bitmezőoffszetet ad meg a $-2^{31}...2^{31}-1$ tartományban. A két eset közüli választást a Do bit végzi el.

Az utasításszót egy újabb szó követi, amely az effektív címen belül a bitmezőt pontosabban meghatározza:

bit	15 14 13 12	11	10 9 8 7 6	5	4 3 2 1 0
	0 0 0 0	Do	offset	Dw	hossz

Dw	0 = a szónak a hosszmezeje egy közvetlenül megadott hosszadatot tartalmaz. 1 = a szónak ez a hosszmezeje annak az adatregiszternek a számát tartalmazza, amelyik a tulajdonképpeni távolsági adatot tartalmazza. Mivel ez az adat a regiszterben a $-2^{31}...2^{31}-1$ tartományt foghatja át, ennek a regisztertartalomnak csak a modulo 32-es értéke kerül effektív hosszként felhasználásra.
hossz	ez a mező vagy egy 0...31 közé eső állandót, vagy annak az adatregiszternek a számát tartalmazza, amely a hosszra vonatkozó adatot tartalmazza. A két eset közüli választást a Dw bit végzi el.

Az utasítás leírása:

E művelet végrehajtásakor először felülvizsgálatra kerül a célként megadott bitmező a nullával való egyezőségre és a negativitásra vonatkozóan, hogy a kapcsolókat ennek megfelelően lehessen állítani. Ezt követően a bitmező valamennyi eleme magasra állítódik.

Mind a bitmező kezdete, mind annak hossza megadható adatregiszterekben, vagy közvetlenül is.

Ha a hosszoperandus modulo 32-t nullának adjuk meg, akkor ez 32 bit hosszúságnak felel meg.

Ez a művelet csak a 68020-as processzorba van beépítve.

bitmező-operandus vizsgálata
Bit Field TeST
CSAK A 68020-AS PROCESSZORNÁL

Művelet	**Assembler szintaxis**
BFTST "ea" Operandus hossza:	BFTST offset: hossz bitmező

Kapcsolók:

X	N	Z	V	C
–	*	*	0	0

X nem változik.
N értéke 1 lesz, ha a vizsgálandó bitmező legnagyobb helyiértékű bitje magasra volt állítva, egyébként törlődik.
Z értéke 1 lesz, ha a vizsgálandó bitmező zérus volt, egyébként törlődik.
V törlődik.
C törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	0	1	0	0	0	1	1	effektív cím					

effektív cím:	megadja azt a célként szolgáló kettős-szót, amelyben a bitmezőt meg kell vizsgálni. A **vezérlés** kategóriák valamennyi címzésmódja megengedett, valamint a Dx címzésmód, tehát az *adatregiszter közvetlen* is.
Do	0 = OFFSET, az abszolút offszetet tartalmazza.
	1 = OFFSET, a 8...6 bitekben annak az adatregiszternek a számát tartalmazza, amelyik a tulajdonképpeni offszetet tartalmazza.

offset	Itt 0 és 31 közé eső közvetlen érték, vagy 0 és 7 közé eső regiszterszám áll, azt az adatregisztert specifikálva, amely aztán egy bitmezőoffszetet ad meg a $-2^{31}...2^{31}-1$ tartományban. A két eset közüli választást a Do bit végzi el.

Az utasításszót egy újabb szó követi, amely az effektív címen belül a bitmezőt pontosabban meghatározza:

bit	15 14 13 12	11	10 9 8 7 6	5	4 3 2 1 0
	0 0 0 0	Do	offset	Dw	hossz

Dw	0 = a szónak a hosszmezeje egy közvetlenül megadott hosszadatot tartalmaz. 1 = a szónak ez a hosszmezeje annak az adatregiszternek a számát tartalmazza, amelyik a tulajdonképpeni távolsági adatot tartalmazza. Mivel ez az adat a regiszterben a $-2^{31}...2^{31}-1$ tartományt foghatja át, ennek a regisztertartalomnak csak a modulo 32-es értéke kerül effektív hosszként felhasználásra.
hossz	ez a mező vagy egy 0...31 közé eső állandót, vagy annak az adatregiszternek a számát tartalmazza, amely a hosszra vonatkozó adatot tartalmazza. A két eset közüli választást a Dw bit végzi el.

Az utasítás leírása:

E művelet végrehajtásakor először felülvizsgálatra kerül a célként megadott bitmező a nullával való egyezőségre és a negativitásra vonatkozóan, hogy a kapcsolókat ennek megfelelően lehessen állítani.

Mind a bitmező kezdete, mind annak hossza megadható adatregiszterekben, vagy közvetlenül is.

Ha a hosszoperandus modulo 32-t nullának adjuk meg, akkor ez 32 bit hosszúságnak felel meg.

Ez a művelet csak a 68020-as processzorba van beépítve.

B K P T ha a processzor felismerte a törésponti vektort (breakpoint-vektor), akkor végrehajtja az átadott utasítást, egyébként ILLEGAL INSTRUCTION TRAP (illegális utasítás megszakítás)

emulátorsegítség
BreakPoinT
CSAK A 68010-ESTŐL

Művelet	**Assembler szintaxis**
BKPT "adat"	BKPT # "adat"
Operandus hossza:	implicit három bit

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	1	0	0	0	0	1	0	0	1	vektor		

vektor: 0 és 7 közötti érték, amelyet törésponti vektorként a címbuszra kell helyezni (csak 68020-as).

Az utasítás leírása:

Ez a műveleti kód (OP–Code) egy törésponti-ciklust indít el.
A következő utasítás végrehajtása csak a törésponti-ciklus nyugtázása után kezdődik meg.

E ciklus során a csatlakoztatott perifériáról csak egy utasításszó vihető a buszra, és adható az nDSACKx jellel tovább (csak a 68020-asnál!).
Ha ez nem, vagy nem helyesen történik meg, vagy nem 68020-as van a rendszerben, akkor a processzor olyan kizárási feldolgozást indít el, mint amilyet egy illegális utasítás felismerésekor (lásd ehhez a megfelelő hadverfejezetet).

B R A PC + címkülönbség – : PC

relatív ugrás
BRAnch

Művelet	**Assembler szintaxis**
BRA "címke"-hez	BRA "címke"
Operandus hossza:	byte vagy szó

Kapcsolók

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	1	0	0	0	0	0	8 bites címkülönbség							

Az utasításszót egy újabb szó követi, ha a 8 bites címkülönbség nulla. Ebben az esetben egy „távoli" relatív ugrásról van szó, amelynek a különbségi értékét ez a második szó tartalmazza.

Az utasítás leírása:

A PC programszámláló értéke a megadott címkülönbség értékével megváltozik. Ha a címkülönbség – az ugróutasítást követő első byte-tól számítva – a –128...+127 tartományon belül van, akkor a különbséget magában az utasításban implicite meglevő 8 bites érték adja meg. Ha az ugrás távolsága –32 768...32 767 között van, akkor ez egy „távoli" ugrás, és a különbség értéke az utasításszót követő szóban van elhelyezve. Ebben az esetben az implicit 8 bites érték mindig nulla, és az ugrási cím kiszámítása az ugrástávolságot megadó szó utáni címtől kezdődik.

B S E T

$!(\text{cél}_{\text{bitszám}}) -: Z$
$1 -: \text{cél}_{\text{bitszám}}$

egy bit vizsgálata és magasra állítása
Bit test and SET

Művelet	**Assembler szintaxis**
BSET Dx, "ea"	**BSET Dx, "ea"**
BSET "adat", "ea"	**BSET # "adat", "ea"**
Operandus hossza:	**byte vagy kettős-szó**

Kapcsolók

X	N	Z	V	Z
–	–	*	–	–

X nem változik.
N nem változik.
Z értéke 1 lesz, ha a vizsgálandó bit értéke nulla volt, egyébként törlődik.
V nem változik.
C nem változik.

Az utasítás formátuma, ha a bit száma a regiszterben van:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	Dx			1	1	1	effektív cím					

Dx	az az adatregiszter, amelyik az illető bit számát tartalmazza.
effektív cím:	meghatározza a céloperandust; az **adatok * változtatható** kategóriák valamennyi címzésmódja használható.

Az utasítás leírása:

Ennek az utasításnak a feldolgozásakor először átmásolódik a zéró-kapcsolóba annak a bitnek a komplemense, amelyet a Dx vagy egy közvetlen érték, és az effektív cím ad meg. Ezután az előzőleg olvasott helyre egy 1-es kerül visszaírásra.

Ha a cél címe a nyolc adatregiszter valamelyikében van, akkor bitmaszkként a megadott bitszám modulo 32-es értékét használjuk, egyébként, tehát ha a tárban levő valamelyik bitet vizsgáljuk, akkor a megadott bitszám modulo 8-as értékét dolgozzuk fel. Erre a megkülönböztetésre azért van szükség, hogy egyrészt egy CPU-regiszterben egy komplett, 32 bit szélességű kettős-szó feldolgozható legyen, másrészt viszont azért, hogy különleges buszkoncepciók megvalósításához akár egyetlenegy, tárbeli byte-ot is vizsgálni lehessen.
Ha a bitszám közvetlen értékként adott, akkor ez a műveleti kódot követő első szó legalsó 8 bitjén helyezkedik el. A BTST utasítás leírását figyelembe kell venni!

Az utasítás formátuma, hogy a bitszám közvetlen érték:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	1	0	0	0	1	1	effektív cím					

B S R

PC –: –(ind. SP)
PC + indulóérték –: PC

relatív ugrás egy subrutinra (alprogramra)
Branch to SubRoutine

Művelet	**Assembler szintaxis**
BRA "címke"	BRA "címke"
Operandus hossza:	byte vagy szó

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	1	0	0	0	0	1	8 bites címkülönbség							

Az utasításszót egy újabb szó követi, ha a 8 bites címkülönbség nulla. Ebben az esetben egy „távoli" relatív ugrásról van szó, amelynek a különbségi értékét ez a második szó tartalmazza.

Az utasítás leírása:

Először ezt az utasítást követő műveleti kód címe a verembe kerül, majd a PC programszámláló a megadott címkülönbséggel megváltozik. Ha a címkülönbség – az ugró-utasítást követő első byte-tól számítva – a −128...+127 tartományon belül van, akkor a különbséget magában az utasításban implicite meglevő 8 bites érték adja meg. Ha az ugrás távolsága −32 768...32 767 között van, akkor ez egy „távoli" ugrás, és a különbség értéke az utasításszót követő szóban van elhelyezve. Ebben az esetben az implicit 8 bites érték mindig nulla, és az ugrási cím kiszámítása az ugrástávolságot megadó szó utáni címtől kezdődik.

BTST

!(cél$_{\text{bitszám}}$) –: Z

egy bit vizsgálata
Bit TeST

Művelet	**Assembler szintaxis**
BTST Dx, "ea"	BTST Dx, "ea"
BTST "adat", "ea"	BTST# "adat", "ea"
Operandus hossza:	byte vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
–	–	*	–	–

X nem változik.
N nem változik.
Z értéke 1 lesz, ha a vizsgálandó bit zérus volt, egyébként törlődik.
V nem változik.
C nem változik.

Az utasítás formátuma, ha a bitszám a regiszterben van:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	Dx			1	0	0	effektív cím					

Dx:	az az adatregiszter, amelyik az illető bitszámot tartalmazza.
effektív cím:	meghatározza a céloperandust; az **„adatok * változtatható”** valamennyi címzésmódja használható.

Az utasítás leírása:

Ennek az utasításnak a feldolgozásakor először átmásolódik a zérókapcsolóban annak a bitnek a komplemense, amelyet a Dx vagy egy közvetlen érték, és az effektív cím ad meg.

Ha a cél címe a nyolc adatregiszter valamelyikében van, akkor bitmaszkként a megadott bitszám modulo 32-es értékét használjuk, egyébként, tehát ha a tárban levő valamelyik bitet vizsgáljuk, akkor a megadott bitszám modulo 8-as értékét dolgozzuk fel. Erre a megkülönböztetésre itt nincs szükség, de azért, hogy a BCHG, BSET és BCLR utasításokkal kód-kompatíbilis legyen, itt is fennállnak ugyanazok a korlátozások.

Ha a bitszám közvetlen értékként van megadva, akkor ez a műveleti kódot követő szó legalsó 8 bitjén van. Érdekességként felhívjuk a figyelmet, hogy a BCLR, BSET és a BTST utasításoknál (a 68000-es és a 68008-as adatlapja szerint) itt egy 16 bites értéket kell tudni megadni. A BCHG utasításnál azonban nem, ennél a legfelső 8 bitnek nullának kell lenni. Ez mindenféle logikának, és főként a 68020-as szoftverleírásának ellene mond. Ajánljuk ezt kísérletező kedvű Olvasóink figyelmébe!

Az utasítás formátuma, ha a bitszám közvetlen érték:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	1	0	0	0	1	1	effektív cím					

CALLM

modulstátus –: (SP)–
cél –: modulstátus

modul meghívása
CALL Modula
CSAK A 68020-AS PROCESSZORNÁL

Művelet	**Assembler szintaxis**
CALLM "modul" "adat"-tal	CALLM# "adat", "ea"
Operandus hossza:	byte vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma, ha a bitszám a regiszterben van:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	1	1	0	1	1	effektív cím					

Közvetlenül az utasításszó után következik egy újabb szó, amelynek a legalsó 8 bitje a közvetlen értéket tartalmazza, a felső 8 bit értéke nulla.

effektív cím: a céloperandust határozza meg; a **vezérlés** kategória valamennyi címzésmódja használható.

Az utasítás leírása:

Ez az utasítás egy modult hív meg, amelynél a cél címe a moduldeszkriptort, az állandó adatok pedig az átadandó paraméterek számát adják meg (byte-okban).
Az új moduladatkészlet a verem pillanatnyi pozícióján készül el.

CAS/CAS2

cél – összehasonlítandó operandus;
ha nulla, új értékű operandus –: cél
egyébként cél –: összehasonlítandó operandus

operandus összehasonlítása és cseréje
Compare And Swap with operand
CSAK A 68020-AS PROCESSZORNÁL

Művelet	**Assembler szintaxis**
CAS "ea"-t Dv-vel, csere Dn	CAS.X Dv, Dn, "ea"
CAS2 Rx1-et Dv1-gyel, csere Dn1 és Rx2-t Dv2-vel, csere Dn2	CAS2.X Dv1 : Dv2, Dn1 : Dn2, Rx1 : Rx2
Operandus hossza:	byte, szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
–	*	*	*	*

X nem változik.
N értéke 1 lesz, ha a különbség negatív, egyébként törlődik.
Z értéke 1 lesz, ha a különbség zérus, egyébként törlődik.
V értéke 1 lesz, ha a különbségnél túlcsordulás lép fel, egyébként törlődik.
C értéke 1 lesz, ha a különbségnél negatív túlcsordulás lép fel, egyébként törlődik.

Az utasítás formátuma **egyszeres operandus (CAS) esetén:**

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	1	hossz		0	1	1	effektív cím					

Közvetlenül az utasításszó után egy újabb szó következik:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	reg. Dn			0	0	0	reg. Dv		

hossz:	az operandus hosszát határozza meg: 00 = byte 01 = szó 10 = kettős-szó
effektív cím:	meghatározza a céloperandust; a **változtatható** kategória valamennyi címzésmódja megengedett.
Dn reg.	annak az adatregiszternek a számát adja meg, amely a cél számára az új értéket tartalmazza.
Dv reg.	annak az adatregiszternek a számát adja meg, amely a céllal való összehasonlításra szolgál.

Az utasítás effektív címként **közvetlen** értékeket is fel tud dolgozni, de a processzor ezt úgy értelmezi, hogy most két, egyenértékű operandust kell szinkron módon feldolgoznia.

Az utasítás formátuma kétszeres operandus (CAS2) esetén:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	1	hossz		0	1	1	1	1	1	1	0	0

Közvetlenül az utasításszó után két újabb szó következik:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	D1	reg. Rx1			0	0	0	reg. Dn1			0	0	0	reg. Dv1		
	D2	reg. Rx2			0	0	0	reg. Dn2			0	0	0	reg. Dv2		

D1/D2	azt határozzák meg, hogy az Rx1/Rx2 adat- vagy címregiszterek-e: 0 = adatregiszter 1 = címregiszter
Rx1/Rx2	mindig annak a regiszternek a számát adják meg, amelyik a céloperandus címét tartalmazza. Ha ez a két operandus a tárban átfedi egymást, akkor a céloperandus esetleges megváltoztatásakor annak állapotára vonatkozóan semmiféle megbizható kijelentést sem lehet tenni.
Dn1/Dn2	azoknak az adatregisztereknek a számát jelölik, amelyek azt az új értéket tartalmazzák, amely – a cél- és az összehasonlítandó operandus egyezőség esetén – a céloperandust felülírja.
Dv1/Dv2	ezek a regiszterszámok azokat az adatregisztereket jelölik, amelyek tartalma a céloperandussal összehasonlításra kerül.

Az utasítás leírása:

Ez az utasítás egy kibővített összehasonlító (compare-) utasítás. Az effektív cím tartalmának a Dv adatregiszterrel való összehasonlítása után két dolog történhet:
Ha a két érték megegyezik, akkor a célba (az effektív cím) az új értékű, Dn adatregiszter értéke kerül betöltésre;
Ha a két érték nem egyezik meg, akkor a Dv összehasonlító regiszterbe az effektív cím tartalma töltődik.
A CAS utasításnál byte-ok, szavak és kettős-szavak hasonlíthatók össze egymással, a CAS2 utasításnál két érték vehető egyidejűleg figyelembe.
A buszkezelést illetően ez az utasítás a feldolgozása során nem szakítható meg, így ez kifejezetten a többprocesszoros berendezések szinkronizálásra alkalmas utasítás.

C H K Dx kisebb mint 0 vagy nagyobb mint "ea" – 6-os kizárás

regiszter vizsgálata, hogy két határon belül van-e
CHeck register against bounds

Művelet	**Assembler szintaxis**
CHK Dx-et "ea"-hoz képest	CHK.X "ea", Dx
Operandus hossza:	szó, a 68020-asnál kettős-szó is

Kapcsolók:

X	N	Z	V	C
–	*	U	U	U

X nem változik.
N értéke 1 lesz, ha Dx kisebb mint 0, törlődik, ha Dx nagyobb mint "ea"; egyébként meghatározhatatlan.
Z meghatározhatatlan.
V meghatározhatatlan.
C meghatározhatatlan.

Az utasítás formátuma:

bit	15 14 13 12	11 10 9	8 7 6	5 4 3 2 1 0
	0 1 0 0	Dx	hossz	effektív cím

Dx:	az az adatregiszter, amelynek a tartalmát meg kell vizsgálni.
effektív cím:	a céloperandust határozza meg; az **adatok** kategória valamennyi címzésmódja használható.
hossz:	a 68000-es, 68008-as, 68010-es és a 68012-es processzoroknál szóhossz (110), a 68020-asnál adott esetben kettős-szó hossz (100) is lehet.

Az utasítás leírása:

Ez az utasítás arra szolgál, hogy egy adatregiszter alsó szavát két határra vonatkozóan megvizsgálja. Az alsó határ mindig a nulla, a felső határt egy effektív cím specifikálja, itt ennek a 16 bites tartalmát kettes komplemensként kell tekinteni. Ez azt jelenti, hogy a 0...32 767 számtartomány vizsgálható meg. Ha a megadott adatregiszter alsó szava ezeken a határokon kívül van, akkor egy szoftveroldali megszakítás (TRAP) következik be a 6-os vektorra, tehát külön erre a célra lefoglalt CHK–TRAP-ra. Egyéb esetben a kapcsoló módosításán kívül nem történik semmi.

C H K 2

Rx kisebb (LOWER)
vagy Rx nagyobb (HIGHER) – 6-os kizárás

regiszter vizsgálata, hogy két határon belül van-e
CHecK register against bounds
CSAK A 68020-AS PROCESSZORNÁL

Művelet	**Assembler szintaxis**
CHK2 Rx-et "ea"-hoz képest	CHK.X "ea", Rx
Operandus hossza:	byte, szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
–	U	*	U	*

X nem változik.
N nincs meghatározva.
Z értéke 1 lesz, ha a vizsgálandó operandus a két határral megegyezik, egyébként törlődik.
V nincs meghatározva.
C értéke 1 lesz, ha Rx a határokon kívül van, egyébként törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	hossz		0	1	1	effektív cím					

Közvetlenül az utasításszó után egy újabb szó következik, amely az összehasonlítandó regisztert specifikálja:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	DA	Rx. reg.			1	0	0	0	0	0	0	0	0	0	0	0

hossz	meghatározza az operandus hosszát: 00 = byte 01 = szó 10 = kettős-szó
effektív cím:	a céloperandust határozza meg; a **vezérlés** valamennyi címzésmódja használható.
DA	meghatározza, hogy a következő, Rx regiszter cím- vagy adatregiszter-e: 0 = adatregiszter 1 = címregiszter
Rx. reg.	annak az adat-, ill. címregiszternek a számát tartalmazza, amelynek a határait meg kell vizsgálni.

Az utasítás leírása:

Ezzel az utasítással egy cím- vagy adatregiszter két határra vonatkozó vizsgálata végezhető. Az alsó határ az effektív cím által megadott helyen van, a felső határ az operandus hosszával eltolva, mellette.
Ha a megadott cím- vagy adatregiszter az effektív cím által specifikált határokon belül van, akkor a kapcsolók megfelelő beállításán kívül nem történik semmi. Ha a regiszter tartalma a határokon kívül esik, akkor szoftveroldali megszakítás következik be, a 6-os számú vektorra.
Ahhoz, hogy a határvizsgálat előjelhelyes legyen, a matematikailag kisebb értéknek az alacsonyabb határnak, a logikai határvizsgálatnál ennek megfelelően a logikailag kisebb értéknek az alsó határnak kell megfelelnie.
Az adatregisztereknél a határvizsgálatok byte-, szó- vagy kettős-szó szélességekre végezhetők el, a címregisztereknél a szó- vagy byte-összehasonlításoknál ez az érték előjelhelyesen 32 bitre bővül, hogy azután a címregisztert ezzel lehessen összehasonlítani.
Ha a felső és az alsó határ azonos, a vizsgálandó terület akkor tekintendő a határokon belülinek, ha az megegyezik a határokkal.

törlés
CLeaR

Művelet	**Assembler szintaxis**
CLR "ea"	CLR.X "ea"
Operandus hossza:	byte, szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
–	0	1	0	0

X nem változik.
N törlődik.
Z magasra van állítva.
V törlődik.
C törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	0	0	1	0	hossz		effektív cím					

hossz három értéket vehet fel:

00 = byte-művelet
01 = szóművelet
10 = kettős-szó művelet

effektív cím: a céloperandust határozza meg; az **adatok * változtatható kategóriák** valamennyi címzésmódja megengedett.

Az utasítás leírása:

Ez az utasítás a megadott célt törli, és ennek megfelelően módosítja a kapcsolókat.

összehasonlítás
CoMPare

Művelet	**Assembler szintaxis**
CMP "ea"-t Dx-szel	CMP.X "ea", Dx
Operandus hossza:	byte, szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
–	*	*	*	*

X nem változik.
N értéke 1 lesz, ha az eredmény negatív, egyébként törlődik.
Z értéke 1 lesz, ha az eredmény nulla, egyébként törlődik.
V értéke 1 lesz, ha túlcsordulás jön létre, egyébként törlődik.
C értéke 1 lesz, ha negatív átvitel („kölcsönvétel") jön létre, egyébként törlődik.

Az utasítás formátuma:

bit	15 14 13 12	11 10 9	8 7 6	5 4 3 2 1 0
	1 0 1 1	reg.	módus	effektív cím

reg.	célként a nyolc adatregiszter egyikének a számát tartalmazza.
módus	az operandus hosszát határozza meg: byte szó kettős-szó 000 001 010
effektív cím:	a második (forrás-) operandus számára adja meg a címzési módot: egy kivétellel az összes címzésmód megengedett; ha az operandus byte hosszúságú, akkor az An (címregiszter közvetlen) címzésmód nem megengedett.

Az utasítás leírása:

Az utasítás hatására a forrásoperandus tartalma kivonódik a céloperandus tartalmából, de az eredményt nem írjuk be, hanem csak a kapcsolók állítódnak be az eredménynek megfelelően.

címregiszter összehasonlítás
CoMPare Addressregister

Művelet	**Assembler szintaxis**
CMPA "ea"-t Ax-szel	CMPA.X "ea", Ax
Operandus hossza:	szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
–	*	*	*	*

X nem változik.
N értéke 1 lesz, ha az eredmény negatív, egyébként törlődik.
Z értéke 1 lesz, ha az eredmény nulla, egyébként törlődik.
V értéke 1 lesz, ha túlcsordulás keletkezik, egyébként törlődik.
C értéke 1 lesz, ha egy negatív túlcsordulás keletkezik („kölcsönvétel"), egyébként törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	1	1	reg.			módus			effektív cím					

reg.	a nyolc adatregiszter közül annak a számát tartalmazza, amely a mindenkori cél.
módus	az operandus hosszát határozza meg: 011 = szó-operandus. Ahhoz, hogy a cím kiszámítását helyesen lehessen elvégezni, a forrásoperandus előjelhelyesen 32 bitre bővül ki. 111 = kettős-szó operandus
effektív cím	a forrásoperandust jelöli; valamennyi címzési mód megengedett.

Az utasítás leírása:

A CMPA utasítás kiszámítja a cél és a forrás közötti különbséget, az eredmény nem kerül megőrzésre, és az utasítás végén csak a kapcsolók állítódnak be megfelelően.
A CMPA felépítését tekintve csak a módusában különbözik a CMP-től. Míg a CMP-nél ezek az értékek a x00, x01 és x10, addig a CMPA-nál x11. Így a CMPA-t különleges címzésmódú CMP-ként lehet tekinteni.

CMPI

cél – adatok

összehasonlítás közvetlen értékkel
CoMPare Immediate

Művelet	**Assembler szintaxis**
CMPI "adat"-t "ea"-val	CMPI.X# "adat", "ea"
Operandus hossza:	byte, szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
–	*	*	*	*

X nem változik.
N értéke 1 lesz, ha az eredmény negatív, egyébként törlődik.
Z értéke 1 lesz, ha az eredmény nulla, egyébként törlődik.
V értéke 1 lesz, ha túlcsordulás keletkezik, egyébként törlődik.
C értéke 1 lesz, ha egy negatív túlcsordulás keletkezik, egyébként törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	1	1	0	0	hossz		effektív cím					

Az utasításszót egy szó követi, ha az operandus hossza byte vagy szó, és egy kettős-szó követi, ha az operandus hossza kettős-szó. Ha az operandus hossza byte, akkor a szónak csak az alsó 8 bitje (0...7. bit) szignifikáns.

hossz:	három értéket vehet fel: 00 = byte-művelet 01 = szóművelet 10 = kettős-szó művelet
effektív cím	a céloperandust határozza meg; csak az **adatok * változtatható** kategóriák címzésmódjai használhatók.

Az utasítás leírása:

Ez az utasítás egy megadott célból kivon binárisan egy állandót, az eredményt nem őrzi meg, hanem csak a kapcsolókat állítja be ennek megfelelően. Az állandó a tárban közvetlenül az utasításszó után áll, és lehet byte, szó, vagy kettős-szó hosszúságú. A byte hosszúságú operandusok szóhosszúságban foglalnak el címeket a tárban.

összehasonlítás a táron belül
CoMPare in Memory

Műveletek	**Assembler szintaxis**
CMPM (ind.)Ax+ és (ind.)Ay+	CMPM.X (Ax)+, (Ay)+
Operandus hossza:	byte, szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
–	*	*	*	*

X nem változik.
N értéke 1 lesz, ha az eredmény negatív, egyébként törlődik.
Z értéke 1 lesz, ha az eredmény nulla, egyébként törlődik.
V értéke 1 lesz, ha túlcsordulás keletkezik, egyébként törlődik.
C értéke 1 lesz, ha egy negatív túlcsordulás („kölcsönvétel") keletkezik, egyébként törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11 10 9	8	7 6	5	4	3	2 1 0
	1	0	1	1	Ay reg.	1	hossz	0	0	1	Ax reg.

Ay reg.	annak a címregiszternek a száma, amelyik a célregiszter.
hossz	három értéket vehet fel: 00 = byte-művelet 01 = szóművelet 10 = kettős-szó művelet
Ax reg.	hasonlóan a célregiszterhez, annak a címregiszternek a száma, amelyik a forrásregiszter.

Az utasítás leírása:

Ez az utasítás kivonja a célregiszterben levő byte-ból, szóból, kettős-szóból a forrásregiszterben levő byte-ot, szót, kettős-szót. Az eredményt nem őrzi meg, csak a kapcsolók állítódnak be ennek megfelelően. A különbség képzése után mind a cél-, mind a forrásregiszter mutatója (pointer) inkrementálódik.

C M P 2

Rx kisebb (LOWER)
vagy Rx nagyobb (HIGHER)

regiszter vizsgálata, hogy két határon belül van-e
CoMPare register against 2 bouds
CSAK A 68020-AS PROCESSZORNÁL

Művelet	**Assembler szintaxis**
CMP2 Rx-et "ea"-hoz képest	CMP2.X "ea", Rx
Operandus hossza:	byte, szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
–	U	*	U	*

X nem változik.
N nincs meghatározva.
Z értéke 1 lesz, ha a vizsgálandó operandus éppen a két határon van, egyébként törlődik.
V nincs meghatározva.
C értéke 1 lesz, ha Rx a határokon kívül van, egyébként törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	hossz		0	1	1	effektív cím					

Közvetlenül az utasításszó után egy újabb szó következik, amelyik az összehasonlítandó regisztert és az utasítást specifikálja:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	DA	Rx. reg.			0	0	0	0	0	0	0	0	0	0	0	0

hossz	az operandus hosszát határozza meg: 00 = byte 01 = szó 10 = kettős-szó
effektív cím:	a céloperandust határozza meg; a **vezérlés** kategória valamennyi címzésmódja használható.
DA	azt határozza meg, hogy az ezt követő Rx regiszter cím- vagy adatregiszter-e: 0 = adatregiszter 1 = címregiszter
Rx. reg.	annak az adat-, ill. címregiszternek a számát tartalmazza, amelynek a határait vizsgálni kell.

Az utasítás leírása:

Ez az utasítás azt a célt szolgálja, hogy egy cím- vagy adatregisztert két határra vonatkozóan megvizsgáljunk, és a kapcsolókat a vizsgálat eredményének megfelelően állítsuk. Az alsó határ az effektív cím által megadott helyen van, a felső határ az operandus hosszával eltolva, mellette.
Ahhoz, hogy a határvizsgálat előjelhelyes legyen, a matematikailag kisebb értéknek az alacsonyabb határnak, a logikai határvizsgálatnál ennek megfelelően a logikailag kisebb értéknek az alsó határnak kell megfelelnie.
Az adatregisztereknél a határvizsgálatok byte-, szó- vagy kettős-szó szélességekre végezhetők el, a címregisztereknél a szó- vagy byte-összehasonlításoknál ez az érték előjelhelyesen 32 bitre bővül, hogy azután a címregisztert ezzel lehessen összehasonlítani.
Ha a felső és az alsó határ azonos, akkor a vizsgálandó terület akkor tekintendő a határokon belül, ha az megegyezik a határokkal.

c p B_{cc} ha cp_{cc} érvényes, akkor PC + címkülönbség – : PC

relatív ugrás, ha a társprocesszor-feltétel(ek) teljesül(nek)
Branch if CoProcessor-Condition-Codes true
CSAK A 68020-AS PROCESSZORNÁL

Művelet	**Assembler szintaxis**
cpB_{cc} Dx a "címkére"	cpB_{cc} Dx, "címke"
Operandus hossza:	szó, kettős-szó

Kapcsolók

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	1	tp. azon.			0	1	L	feltételkód					

Az utasításszót két vagy három újabb szó követi:

bit	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
	opcionális, a társprocesszortól függő bővítés
	szó
	vagy kettős-szó ugráskülönbség

tp. azon.	megadja a megszólítandó társprocesszor (tp.) azonosítási számát.
L	meghatározza az ugráskülönbség hosszát: 0 = szóhossznyi különbség 1 = kettős-szó hossznyi különbség
feltételkód	itt a társprocesszornál megvizsgálandó feltételkód áll, aminek a bitenkénti interpretálásáról itt természetesen semmit sem lehet mondani. A 68020-as megadja a feltételkódot a társprocesszornak, amely válaszként erre a CPU-t ugrásra vagy a program normál folytatására készteti.
opcionális, társprocesszortól függő bővítés	erre ugyanaz vonatkozik, mint amit a feltételkódnál elmondtunk.
Szó- vagy kettős-szó ugráskülönbség	ha az adott feltétel teljesül, akkor a processzor arra kényszerül, hogy a programszámlálója pillanatnyi értékéhez egy offszetet adjon hozzá. Ennek az offszetnek a szélessége az L bittől függően lehet szó vagy kettős-szó, de minden esetben kettes komplemensű egész számként kell interpretálni.

Az utasítás leírása:

Az utasítás végrehajtásának megkezdésekor a processzor a felülvizsgálandó feltételkódot megküldi a társprocesszornak. Ez utóbbi reagálásától függően végez el most a CPU egy közvetett ugrást, vagy a szokásos módon folytatja a munkáját a következő műveleti kóddal.

Ha a feltétel érvényes, tehát logikailag igaz, akkor a PC programszámláló a megadott címkülönbséggel megváltozik. Az ugráskülönbség vagy – a következő utasítástól számított – mínusz 32 768...+32 767 tartományban van, és ekkor a különbség egy, az utasításszót követő szóban kerül elhelyezésre, vagy pedig −2 147 483 650 és 2 147 483 649 között van, és ekkor az utasításszót követő kettős-szóban kerül elhelyezésre.

cpDB$_{cc}$ ha cp$_{cc}$ érvényes, akkor DX − 1 – : Dx
ha Dx ≠ − 1, PC + címkülönbség – : PC

relatív ugrás, ha a társprocesszor-feltétel(ek) teljesül(nek) és a számláló dekrementálás után: ≠ − 1.
CSAK A 68020-AS PROCESSZORNÁL

Művelet	**Assembler szintaxis**
cpDB$_{cc}$ Dx a "címkére"	cpDB$_{cc}$ Dx, "címke"
Operandus hossza:	szó

Kapcsolók

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	1	tp. azon.			0	0	1	0	0	1	Dx reg.		

Az utasításszót három újabb szó követi:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	0	0	0	feltételkód					
	opcionális, a társprocesszortól függő bővítés															
	szó-ugráskülönbség															

tp. azon	megadja megszólítandó társprocesszor azonosítási számát.
Dx reg.	ez a számlálóregiszter száma (csak adatregiszter).
feltételkód	itt a társprocesszornál megvizsgálandó feltételkód áll, aminek a bitenkénti interpretálásáról itt természetesen semmit sem lehet mondani. A 68020-as megadja a feltételkódot a társprocesszornak, amely válaszként erre a CPU-t ugrásra, vagy a program normál folytatására készteti.
opcionális, társprocesszortól függő bővítés	erre ugyanaz vonatkozik, mint amit a feltételkódnál elmondtunk.
Szó-ugrás-különbség	ha az adott feltétel teljesül, akkor a processzor arra kényszerül, hogy a programszámlálója pillanatnyi értékéhez egy offszetet adjon hozzá. Ezt az offszetet kettes komplemensű, egész számként kell interpretálni.

Az utasítás leírása:

Az utasítás végrehajtásának megkezdésekor a processzor a felülvizsgálandó feltételkódot megküldi a társprocesszornak.
Ha ez a társprocesszor reagálása szerint nem érvényes, tehát logikailag hamis, akkor a megadandó adatregiszter tartalma dekrementálódik. Ha ez ezt követően $\neq -1$, akkor a PC programszámláló a megadott címkülönbséggel megváltozik. Az ugrástávolság – a következő utasítástól számítva – $-32\,768$ és $+32\,767$ között van, és a különbség az opcionális társprocesszor-bővítő szó utáni szóban van elhelyezve.
Ha a társprocesszor-feltétel igaz, vagy a regiszter a dekrementálás után $= -1$, akkor a CPU a soron következő utasítást dolgozza fel.

normál vezérlésátadás a társprocesszornak
CoProcessor GENeral Command

Művelet	**Assembler szintaxis**
cpGEN utasítás, adatok	cpGEN "paraméter"

A pontos szintaxist a társprocesszor írja elő.

Az operandus hossza: nem CPU-operandus

Kapcsolók:

X	N	Z	V	C
C	C	C	C	C

X változatlan, ha a társprocesszor nem változtatja meg.
N változatlan, ha a társprocesszor nem változtatja meg.
Z változatlan, ha a társprocesszor nem változtatja meg.
V változatlan, ha a társprocesszor nem változtatja meg.
C változatlan, ha a társprocesszor nem változtatja meg.

Az utasítás formátuma:

bit	15 14 13 12	11 10 9	8 7 6	5 4 3 2 1 0
	1 1 1 1	tp. azon.	0 0 0	effektív cím

Az utasításszót két újabb szó követi:

bit	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
	társprocesszor-vezérlés
	opcionális, társprocesszortól függő bővítés

tp. azon.	megadja a megszólítandó társprocesszor azonosítási számát.
effektív cím	ez a mező egy, a társprocesszoron kívüli, opcionális operandust jelöl meg. A lehetséges címzésmódokat kizárólag a társprocesszor határozza meg.
társprocesszor-vezérlés	itt áll a tulajdonképpeni, a társprocesszornak átadandó vezérszó. Ennek az interpretálása az alkalmazott társprocesszortól függ.
opcionális, a társprocesszortól függő bővítés	erre ugyanaz vonatkozik, mint amit a társprocesszor-vezérlésnél elmondtunk.

Az utasítás leírása:

Ez a vezérszó arra utasítja a CPU-t, hogy az utasításszót követő vezérszót a specifikált társprocesszorhoz továbbítsa, és esetleg egy megadandó effektív címről még adatokat bocsásson a rendelkezésére.

cpRESTORE — egy társprocesszor egy (régi) állapotának visszaállítása

állapotváltoztatás az effektív címen levő új értékekkel
CSAK A 68020-AS PROCESSZORNÁL
PRIVILEGIZÁLT UTASÍTÁS

Művelet	**Assembler szintaxis**
cpRESTORE cím, paraméterlista	cpRESTORE "ea"
Operandus hossza:	nem CPU-operandus

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	1	tp. azon.			1	0	1	effektív cím					

tp azon.	megadja a megszólítandó társprocesszor azonosítási számát.
effektív cím	ez a mező azt a helyet jelöli, ahol a társprocesszor re-inicializálásához szükséges paraméterlista van. A **vezérlés** kategória valamennyi címzésmódja, és az (An)+, azaz az utólagos inkrementálású címzésmód is megengedett.

Az utasítás leírása:

A társprocesszor az effektív címen levő adatokkal (re-)inicializálásra kerül.

állapot biztosítása az effektív címen
CSAK A 68020-AS PROCESSZORNÁL
PRIVILEGIZÁLT UTASÍTÁS

Művelet	**Assembler szintaxis**
cpSAVE cím, paraméterlista	cpSAVE "ea"
Operandus hossza:	nem CPU-operandus

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11 10 9	8	7	6	5 4 3 2 1 0
	1	1	1	1	tp. azon.	1	0	0	effektív cím

tp azon.	megadja a megszólítandó társprocesszor azonosítási számát.
effektív cím	ez a mező azt a helyet jelöli, ahol a társprocesszor paraméterlistáját el kell helyezni. A **vezérlés** kategória valamennyi címzésmódja, és a –(An), tehát az előzetes dekrementálású címzésmód is megengedett.

Az utasítás leírása:

A társprocesszor az állapotát az effektív cím által megadott tárhelyen helyezi el.

cpS_{CC}

ha cp_{CC} igaz, akkor #$FF – : cél
ha nem, akkor #$OO – : cél

egy byte valamennyi bitjének magasra állítása, ha a társprocesszor-feltétel(ek) teljesül(nek)
Set all one if Coprocessor-Condition is true

CSAK A 68020-AS PROCESSZORNÁL

Művelet	**Assembler szintaxis**
cpS_{CC} byte címe	cpS_{CC} "ea"
Operandus hossza:	byte

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	1	tp. azon.			0	0	1	effektív cím					

Az utasításszót két újabb szó követi:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	0	0	0	feltételkód					
	opcionális, a társprocesszortól függő bővítés															

tp. azon.	megadja a megszólítandó társprocesszor azonosítási számát.
feltételkód	itt a társprocesszornál megvizsgálandó feltételkód áll, aminek a bitenkénti interpretálásáról itt természetesen semmit sem lehet mondani. A 68020-as megadja a feltételkódot a társprocesszornak, amely válaszként erre a CPU-t arra készteti, hogy a célt magasra állítsa vagy törölje.
opcionális, a társprocesszortól függő bővítés	erre ugyanaz vonatkozik, mint amit a feltételkódnál elmondtunk.

Az utasítás leírása:

Az utasítás végrehajtásának megkezdésekor a processzor a felülvizsgálandó feltételkódot megküldi a társprocesszornak. Ez utóbbi reagálásától függően a CPU a célbyte-ot vagy magasra állítja (ha a feltétel teljesül), vagy törli.

szoftveroldali megszakítás, ha a társprocesszor-feltétel teljesül
Set all one if CoProcessor-Condition true
CSAK A 68020-AS PROCESSZORNÁL

Művelet	**Assembler szintaxis**
$cpTRAP_{CC}$ esetleges adatok	$cpTRAP_{CC}$# "adat"
Operandus hossza:	byte

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	1	tp. azon.			0	0	1	1	1	1	módus		

Az utasításszót két vagy három újabb szó követi:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	0	0	0	feltételkód					
	opcionális, a társprocesszortól függő bővítés															
	opcionális szó															
	vagy kettős-szó paraméterként a TRAP-hez															

tp azon.	megadja a megszólítandó társprocesszor azonosítási számát.
feltételkód	itt a társprocesszornál megvizsgálandó feltételkód áll, aminek a bitenkénti interpretálásáról itt természetesen semmit sem lehet mondani. A 68020-as megadja a feltételkódot a társprocesszornak, amely válaszként erre a CPU-t arra készteti, hogy a szoftveroldali megszakítást (TRAP) végrehajtsa, vagy ne.
opcionális, a társprocesszortól függő bővítés	erre ugyanaz vonatkozik, mint amit a feltételkódnál elmondtunk.
(kettős)-szó	itt paramétereket lehet átadni a TRAP részére.

Az utasítás leírása:

Az utasítás végrehajtásának megkezdésekor a processzor a felülvizsgálandó feltételkódot megküldi a társprocesszornak. Ez utóbbi reagálásától függően a CPU (ha a feltétel teljesült) elindítja a TRAP 7 kizárást, vagy a soron következő utasítással folytatja a munkáját.

D B cc

ha cc érvényes, akkor Dx-1 – : Dx
ha Dx ≠ –1, akkor PC + címkülönbség – : PC

relatív ugrás, ha a feltétel(ek) teljesül(nek), és a programszámláló a dekrementálás után ≠ –1.

Művelet	**Assembler szintaxis**
DB_{CC} Dx a "címkéhez"	DB_{CC} Dx, "címke"
Operandus hossza:	szó

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	1	feltétel				1	1	0	0	1	Dx reg.		

Az utasításszót egy újabb szó követi, ami a 16 bites címkülönbséget adja meg.

feltétel	itt a később részletezésre kerülő feltételkombinációk egyike áll.
Dx reg.	ez a számlálóregiszter száma (csak adatregiszter).

Az utasítás leírása:

Az utasítás végrehajtásának megkezdésekor felülvizsgálatra kerül az állapotregiszter a megadott $_{CC}$ feltételre vonatkozóan. Ha ez érvényes, tehát logikailag igaz, akkor a megadandó adatregiszter tartalma dekrementálódik. Ha ez ezt követően $\neq -1$, akkor a PC programszámláló a megadott címkülönbséggel megváltozik. Az ugrástávolság (a következő utasítástól számítva) −32 768... ...+32 767 között van, és a távolság az utasításszót követő szóban van elhelyezve. A $_{CC}$ feltételként az alábbi 14 bitkombináció egyike adható meg:

Név	**Kód**	**Jelentés**	**Név**	**Kód**	**Jelentés**
CC	0100	Carry = 0	LS	0011	nem nagyobb
CS	0101	Carry = 1	LT	1101	kisebb
EQ	0111	egyenlő	MI	1011	negatív
GE	1100	nagyobb/egyenlő	NE	0110	nem egyenlő
GT	1110	nagyobb mint 0	PL	1010	pozitív
HI	0010	nagyobb	VC	1000	túlcsordulás = 0
LE	1111	kisebb/egyenlő	VS	1001	túlcsordulás = 1

E feltételek pontosabb leírását, főként a kapcsolók erre vonatkozó állapotát illetően a fejezet elején írtakra utalunk.

osztás az előjel figyelembevételével
DIVide Signed

Művelet	**Assembler szintaxis**
DIVS Dx "ea"-val	DIVS(.W) "ea", Dx
Operandus hossza:	alapértelmezés szerint szó, a 68020-asnál kettős-szó is megengedett.

Kapcsolók:

X	N	Z	V	C
–	*	*	*	0

X nem változik.
N értéke 1 lesz, ha az eredmény (a hányados) negatív, egyébként törlődik.
Z értéke 1 lesz, ha az eredmény nulla, egyébként törlődik; túlcsordulás esetén Z nem meghatározott.
V értéke 1 lesz, ha túlcsordulás keletkezik, egyébként törlődik.
C törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	0	0	Dx reg.			1	1	1	effektív cím					

Dx reg.	a nyolc adatregiszter egyikének a számát tartalmazza célként.
effektív cím	megadja a második (forrás-) operandus számára a címzési módot: csak az **adatok** kategória címzésmódjai a megengedettek.

Az utasítás leírása:

Az utasítás előjelhelyesen elosztja a 32 bit széles céloperandust a 16 bit széles forrásoperandussal. Az eredmény 32 bit széles értékként visszakerül a célregiszterbe, és itt a hányados a szó alsó részébe, az osztás maradéka pedig a szó felső részébe kerül.

Az operandusok ábrázolása, ugyanúgy mint az eredményé, a szokásos kettes komplemensű aritmetikának felel meg; a maradéknak mindig az az előjele mint az osztandónak, kivéve, ha a maradék nulla.

Az osztáskor kétféle hiba léphet fel:

Ha az osztó zérus, akkor a szoftveroldali megszakítás (TRAP) következik be az 5-ös vektorra.

Ha az osztandó kisebb mint az osztó, tehát az osztás eredménye 1-nél kisebb, akkor az ilyen esetben magasra állított túlcsordulás- (V) biten keresztül hibajelzés következik, az operandusok viszont nem változnak meg.

osztás az előjel figyelmen kívül hagyásával
DIVide Unsigned

Művelet	**Assembler szintaxis**
DIVU Dx az "ea"-val	DIVU(.W) "ea", Dx
Operandus hossza:	alapértelmezésben szó, 68020-asnál kettős-szó is megengedett.

Kapcsolók:

X	N	Z	V	C
–	*	*	*	0

X nem változik.
N értéke 1 lesz, ha az eredmény (a hányados) legnagyobb helyiértékű bitje magasra van állítva, egyébként törlődik.
Z értéke 1 lesz, ha az eredmény nulla, egyébként törlődik; túlcsordulás esetén Z nincs meghatározva.
V értéke 1 lesz, ha túlcsordulás keletkezik, egyébként törlődik.
C törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	0	0	Dx reg.			0	1	1	effektív cím					

Dx reg.	a nyolc adatregiszter egyikének a számát tartalmazza célként.
effektív cím	megadja a második (forrás-) operandus számára a címzési módot: csak az **adatok** kategória címzésmódjai a megengedettek.

Az utasítás leírása:

Az utasítás elosztja a 32 bit széles céloperandust a 16 bit széles forrásoperandussal. Az eredmény 32 bit széles értékként visszakerül a célregiszterbe, és itt a hányados a szó alsó részébe, az osztás maradéka pedig a szó felső részébe kerül.

Az osztás minden esetben az előjel figyelmen kívül hagyásával történik.

Az osztáskor kétféle hiba léphet fel:

Ha az osztó nulla, akkor a szoftveroldali megszakítás (TRAP) következik be az 5-ös vektorra.

Ha az osztandó kisebb mint az osztó, tehát az osztás eredménye kisebb 1-nél, akkor az ilyen esetben magasra állított túlcsordulás- (V) biten keresztül hibajelzés következik, az operandusok viszont nem változnak meg.

logikai kizáró-VAGY utasítás
logical Exclusic OR

Művelet	**Assembler szintaxis**
EOR Dx "ea"-val	EOR.X Dn, "ea"
Operandus hossza:	byte, szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
–	*	*	0	0

X nem változik.
N értéke 1 lesz, ha az eredmény legnagyobb helyiértékű bitje magasra van állítva, egyébként törlődik.
Z értéke 1 lesz, ha az eredmény nulla, egyébként törlődik.
V törlődik.
C törlődik.

Az utasítás formátuma:

bit	15 14 13 12	11 10 9	8 7 6	5 4 3 2 1 0
	1 0 1 1	Dx reg.	módus	effektív cím

Dx reg.	a nyolc (forrás-) adatregiszter egyikének a számát tartalmazza.
módus	az operandus hosszát határozza meg:

byte	szó	kettős-szó
000	001	010
100	101	110

effektív cím	megadja a második operandus számára a címzési módot: a **tár * változtatható** kategóriák valamennyi címzésmódja használható.

Az utasítás leírása:

Az utasítás elvégzi a forrás és a cél tartalma közötti, logikai kizárás-VAGY (EOR) műveletet, az eredmény a célban kerül elhelyezésre, és a kapcsolók az eredménynek megfelelően állítódnak be.
A forrásnak mindig egy adatregiszternek kell lennie.

EORI cél EOR adatok – : cél

logikai kizáró-VAGY utasítás közvetlen értékkel
logical Exclusiv OR with Immediate value

Művelet	**Assembler szintaxis**
EORIS "adat" az "ea"-val	**EORI.X # "adat", "ea"**
Operandus hossza:	**byte, szó vagy kettős-szó**

Kapcsolók:

X	N	Z	V	C
–	*	*	0	0

X nem változik.
N értéke 1 lesz, ha az eredmény legmagasabb helyiértékű bitje magasra van állítva, egyébként törlődik.
Z értéke 1 lesz, ha az eredmény nulla, egyébként törlődik.
V törlődik.
C törlődik.

Az utasítás formátuma:

bit	15 14 13 12	11 10 9 8	7 6	5 4 3 2 1 0
	0 0 0 0	1 0 1 0	hossz	effektív cím

Az utasításszót egy szó követi, ha az operandus hossza byte vagy szó, és egy kettős-szó követi, ha az operandus hossza kettős-szó. Ha az operandus byte hosszúságú, akkor az utasításszót követő szónak csak a legalsó 8 bitje (0...7. bit) szignifikáns.

hossz	három értéket vehet fel: 00 = byte-művelet 01 = szó hosszúságú művelet 10 = kettős-szó hosszúságú művelet
effektív cím	a céloperandust határozza meg; az **adatok * változtatható** kategóriák címzésmódjai használhatók, az állapotregiszterrel összefüggésben a **közvetlen** is.

Az utasítás leírása:

Ez az utasítás egy állandó érték és egy megadandó cél között logikai kizáró-VAGY műveletet végez el, és beállítja az illető kapcsolókat. Az állandó érték a tárban közvetlenül az utasításszó után van, hossza lehet byte, szó vagy kettős-szó. A byte hosszúságú operandusok szóhosszúságban helyezkednek el.

Ha a cél az állapotregiszter, akkor ehhez kétféle módon lehet hozzáférni:
- ha byte hosszúságú műveletről van szó, akkor az állapotregiszternek csak a user-fele kerül felhasználásra,
- ha szó hosszúságú műveletről van szó, akkor ez az utasítás egy PRIVILEGIZÁLT UTASÍTÁS.

regisztertartalmak cseréje
EXchanGe register

Művelet	**Assembler szintaxis**
EXG Rx-et Ry-val	EXG Rx, Ry
Operandus hossza:	kettős-szó

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	0	0	Rx reg.			1	módus					Ry reg.		

Rx reg.	vagy egy adat- vagy egy címregiszter számát tartalmazza, a módusbitek értékétől függően. Ha a csere egy adat- és egy címregiszter között történik meg, akkor ez a mező mindig az adatregiszterre vonatkozik.
módus	azt adja meg, hogy milyen regiszterek között történik meg a csere: 01000 = adatregiszter : – : adatregiszter 01001 = címregiszter : – : címregiszter 10001 = adatregiszter : – : címregiszter

Ry reg.	a második adat- vagy címregiszter számát tartalmazza, a módusbitek értékétől függően. Ha a csere egy adat- és egy címregiszter között történik meg, akkor ez a mező mindig a címregiszterre vonatkozik.

Az utasítás leírása:

Az utasítás két, megadandó regiszter tartalmát egymással kicseréli. Ennél a műveletnél mindig kettős-szó hosszúságú operandusok szerepelnek.
A csere adat- és címregiszterek, ill. ezek mindenféle kombinációja között lehetséges.

EXT/EXTB cél (előjeles) – : cél
EXTB csak a 68020-asnál!

operandus bővítése előjelhelyesen
Sign EXTend

Művelet	**Assembler szintaxis**
EXT Dx	EXT.X Dx
EXTB Dx	EXTB.L Dx
Operandus hossza:	szó vagy kettős-szó 68020-asnál byte is

Kapcsolók

X	N	Z	V	C
–	*	*	0	0

X nem változik.
N értéke 1 lesz, ha az eredmény negatív, egyébként törlődik.
Z értéke 1 lesz, ha az eredmény nulla, egyébként törlődik.
V törlődik.
C törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	1	0	0	módus			0	0	0	Dx reg.		

módus	megadja az operandus hosszát: 010 = a cél legalsó byte-jából előjeles bővítéssel szó hosszúságú operandus lesz. 011 = a cél legalsó szavából előjeles bővítéssel kettős-szó hosszúságú operandus lesz. 111 = a cél legalsó byte-jából előjeles bővítéssel kettős-szó hosszúságú operandus lesz (csak a 68020-asnál!).
Dx reg.	itt a bővítendő adatregiszter száma áll.

Az utasítás leírása:

Az operandus kétszeres (a 68020-asnál akár négyszeres) bitszélességűre bővül. Ennek során a bővítendő operandus legmagasabb helyiértékű bitje a fölötte álló bitekbe másolódik. Szóhosszúság esetén tehát a 7. bit a 8...15 bitekbe, kettős-szó hosszúság esetén a 15. bit a 16...31 bitekbe, byte-hosszúság esetén – a 68020-asnál – a 7. bit a 8...31 bitekbe másolódik.
A cél mindig valamelyik adatregiszter.

ILLEGAL

PC – : -(SSP)
SR – : -(SSP)
vektor$_4$ – : PC

nem megengedett utasítás
ILLEGAL instruction

Művelet	**Assembler szintaxis**
ILLEGAL	nincs
Operandus hossza:	nincs operandus

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	1	0	1	0	1	1	1	1	1	1	0	0

Az utasítás leírása:

Ha a processzor a program végrehajtása során ezt a műveleti kódot olvassa, akkor elindul egy ILLEGÁLIS UTASÍTÁS kizárásnak a feldolgozása. A processzor a programszámláló és az állapotregiszter pillanatnyi tartalmát tárolja a verembe, majd ezután a 4-es vektorra ugrik.
A 68000-esnél és a 68008-asnál néhány további műveleti kód nincs dekódolva, ezek a későbbi processzoroknál – főként a többprocesszoros rendszerek szinkronizálásához – új utasítások lesznek.

abszolút ugrás
JuMP absolute

Művelet	**Assembler szintaxis**
JMP "ea"-ra	JMP "ea"
Operandus hossza:	implicit, cél címe = kettős-szó

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	1	1	1	0	1	1	effektív cím					

effektív cím	a cél címét adja meg. A **vezérlés** kategória valamennyi címzésmódja megengedett.

Az utasítás leírása:

Abszolút ugrás az effektív cím által megadott ugrási címre.

JSR

PC – : -(xSP)
cél – : PC

abszolút ugrás szubrutinra (alprogramra)
Jump to SubRoutine absolute

Művelet	**Assembler szintaxis**
JSR "ea"-ra	JSR "ea"
Operandus hossza:	implicit, cél címe = kettős-szó

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	1	1	1	0	1	0	effektív cím					

effektív cím	a cél címét adja meg. A **vezérlés** kategória valamennyi címzésmódja megengedett.

Az utasítás leírása:

Miután a programszámláló pillanatnyi értéke tárolásra került a verembe (ami ezen JSR utasítást követő utasítás címe), abszolút ugrás történik az effektív cím által megadott ugrási címre.

abszolút cím töltése címregiszterbe
Load Effective Address to Addressregister

Művelet	**Assembler szintaxis**
LEA "ea"-t Ax-be	LEA "ea", Ax
Operandus hossza:	kettős-szó

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	Ax reg.			1	1	1	effektív cím					

Ax reg.	célként a nyolc címregiszter (Ax) egyikének a számát tartalmazza.
effektív cím	a **vezérlés** kategória valamennyi címzésmódja megengedett.

Az utasítás leírása:

Az effektív címen levő kettős-szó hosszúságú információ betöltésre kerül a megadandó címregiszterbe.

LINK

Ax – : -(SP)
SP – : Ax
SP + "Localstack" – : SP

verem előkészítése paraméterátadáshoz
LINK local basepointer

Művelet	**Assembler szintaxis**
LINK Ax-hez "Localstack"-kel	LINK Ax, # "Localstack"
Operandus hossza:	alapértelmezés (default)

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	1	1	1	0	0	1	0	1	0	Ax reg.		

Közvetlenül a műveleti kód után egy újabb szó következik, amely a közvetlen címoffszetet adja meg a veremmutatónak.

Ax reg.	a nyolc címregiszter egyikének a számát tartalmazza, amely lokális bázisként kerül felhasználásra.

Az utasítás leírása:

Az utasítás végrehajtása során egyidejűleg több művelet kerül elvégzésre: elsőként a megadott címregiszter tartalma a szokásos módon a verembe kerül.

Ezután a veremmutató pillanatnyi értéke átmásolódik az éppen tárolt regiszterbe, és a veremmutató azzal az értékkel változik meg, amelyik a műveleti kódot követő szóban áll. Egyedi utasításokként ezekre példa:

```
MOVE.L     Ax,-(SP)
MOVEA.L    SP,Ax
ADDA.L     SP, # "Localstack"
```

Ennek az utasításnak a sokoldalú alkalmazási lehetőségeivel a 7. fejezetben foglalkozunk részletesen.

LSL / LSR cél eltolva –: cél

egy regiszter bitenkénti logikai eltolása balra/jobbra
Logical Shift Left/Right

Művelet	**Assembler szintaxis**
LSx Dx-et Dy hellyel	LSx.X Dy,Dx
LSx Dx-et "adat" hellyel	LSx.X #"adat", Dx
LSx "ea"-t 1 hellyel	LSx.X "ea"
Operandus hossza:	byte, szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
*	*	*	0	*

X az operandus legutolsó kitolt bittel egyezik meg. Ha a helyek száma nulla, akkor X nem változik.
N értéke 1 lesz, ha az eredmény legnagyobb helyiértékű bitje magas, egyébként törlődik.
Z értéke 1 lesz, ha az eredmény nulla, egyébként törlődik.
V törlődik.
C az operandusból legutoljára kitolt bittel egyezik meg. Ha a helyek száma nulla, akkor C törlődik.

Az utasítás formátuma egy adatregiszter eltolásához:

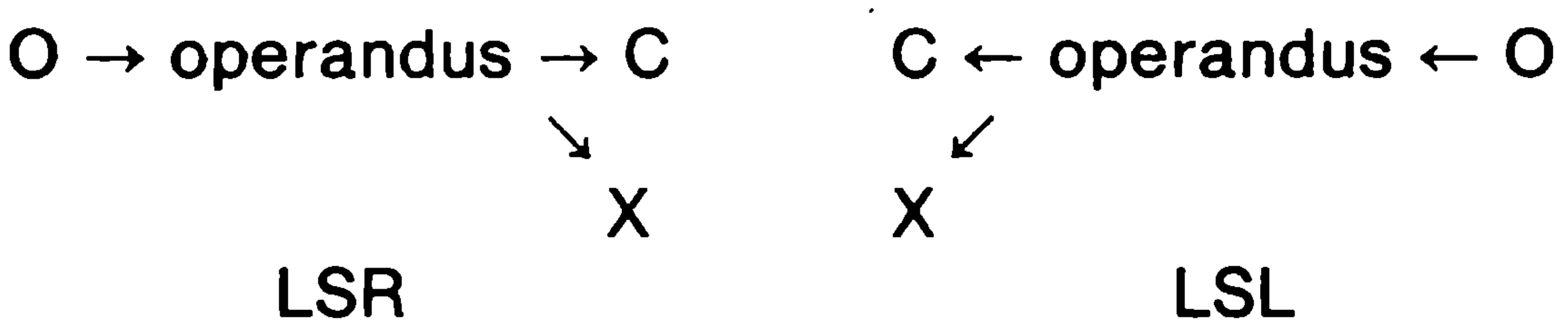
bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	0	reg.			dr	hossz		i	0	1	regiszter		

reg.	a nyolc adatregiszter (Dy) közül annak a számát tartalmazza, amelyik az eltolások számát, ill. az abszolút értéket („data") adja meg (az i bittől függően!). Egy esetleges abszolút érték számszerűen 0 és 7 között (3 bit) lehet, a 0 érték nyolc eltolásnak felel meg.
dr	ez a bit az eltolás irányát (DiRection) adja meg: 0 = eltolás jobbra 1 = eltolás balra

hossz	három értéket vehet fel: 00 = byte hosszúságú művelet 01 = szó hosszúságú művelet 10 = kettős-szó hosszúságú művelet
i	azt határozza meg, hogy a reg. egy regisztert vagy egy abszolút értéket jelöl-e: 0 = abszolút érték 1 = adatregiszter
regiszter	ez a három bit azt az adatregisztert adja meg, amelynek a tartalmát el kell tolni (a cél címe).

Az utasítás leírása:

Ennek az utasításnak a hatására a megadandó cél tartalma egy, ugyancsak megadandó számú bináris hellyel jobbra, ill. balra tolódik.
Az eltolás irányát eltérő assembler szintaxis jelöli (LSL/LSR), a két esetben létrehozott kód azonban csak egy bitben különbözik egymástól.
A cél címeként egy adatregiszter adható meg. Ebben az esetben a cél címe (tehát az adatregiszter) mellett még meg kell adni az eltolandó helyek számát. Ennek megadása vagy konstans formájában, vagy egy másik adatregiszterben lehetséges.
Ha viszont célként nem specifikálunk adatregisztert, akkor célként a **tár * változtatható** valamennyi címzésmódja megengedett, de az eltolandó helyek száma szabadon nem adható meg. Az ebben az esetben mindig 1.
Továbbá arra kell ügyelni, hogy az operandus hosszának megválasztása (byte, szó vagy kettős-szó) csak regisztereltolás esetén lehetséges. A tárra irányuló művelet esetén az operandus hossza mindig szó.



Az utasítás formátuma a tárbeli eltoláshoz

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	0	0	0	1	dr	1	1	effektív cím					

dr	az a bit az eltolás irányát (DiRection) adja meg: 0 = eltolás jobbra 1 = eltolás balra
effektív cím	a céloperandus határozza meg; az **adatok * változtatható** valamennyi címzésmódja megengedett.

adatok átvitele a forrásból a célba
MOVE sourcedata to destination

Művelet	**Assembler szintaxis**
MOVE "ea"-t "ea"-ba	MOVE.X "ea", "ea"
Operandus hossza:	byte, szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
–	*	*	0	0

X nem változik.
N értéke 1 lesz, ha az eredmény negatív, egyébként törlődik.
Z értéke 1 lesz, ha az eredmény nulla, egyébként törlődik.
V törlődik.
C törlődik.

Az utasítás formátuma:

bit	15 14	13 12	11 10 9	8 7 6	5 4 3	2 1 0
	0 0	hossz	y módus	reg. y	reg. x	x módus

hossz	három értéket vehet fel: 00 = byte hosszúságú művelet 01 = szó hosszúságú művelet 10 = kettős-szó hosszúságú művelet
reg. y y módus	ez a két adat határozza meg a cél címét, az **adatok * változtatható** kategóriák valamennyi címzésmódja megengedett.
reg. x x módus	ez a forrás effektív címmezeje. Valamennyi címzésmód megengedett, kivéve azt az esetet, ha byte-operandussal dolgozunk: ilyenkor az An (adatregiszter közvetlen) nem megengedett.

Az utasítás leírása:

Az adatok a forráshelyről átkerülnek a célba, miközben néhány kapcsoló változhat. A forrás tartalma nem változik.

adatok átvitele a forrásból a feltételkód-regiszterbe
MOVE sourcedata to Condition-Code-Register

Művelet	**Assembler szintaxis**
MOVE "ea"-t a CCR-be	MOVE "ea", CCR
Operandus hossza:	szó

Kapcsolók:

X	N	Z	V	C
*	*	*	*	*

X a forrásoperandusnak megfelelően állítódik.
N a forrásoperandusnak megfelelően állítódik.
Z a forrásoperandusnak megfelelően állítódik.
V a forrásoperandusnak megfelelően állítódik.
C a forrásoperandusnak megfelelően állítódik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	0	1	0	0	1	1	effektív cím					

effektív cím	a forrásoperandust jelöli; az **adatok** kategória valamennyi címzésmódja megengedett.

Az utasítás leírása:

Az adatok a forráshelyről átkerülnek a feltételkód-regiszterbe, tehát a kapcsolók meghatározott értékre állítódnak. A forrás tartalma nem változik meg. Bár a forrásoperandus műszaki okokból szó hosszúságú, a célnak csak az alacsonyabb helyiértékű szava kerül átírásra. A komplett állapotszó felülírását a MOVE to SR utasítás végzi el, ami privilegizált.

adatok átvitele a feltételkód-regiszterből
MOVE Condition-Code Register
CSAK A 68010-ES PROCESSZORTÓL KEZDŐDŐEN

Művelet	**Assembler szintaxis**
MOVE CCR-t "ea"-ba	MOVE CCR, "ea"
Operandus hossza:	szó

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	0	1	0	0	1	1	effektív cím					

effektív cím	a forrásoperandust jelöli; az **adatok * változtatható** kategóriák valamennyi címzésmódja megengedett.

Az utasítás leírása:

Az állapotregiszter user részének a tartalma az effektív címre töltődik. Az utasítás ilyen formában csak a 68010-es processzortól kezdődően értelmezett.

adatok átvitele a forrásból az állapotregiszterbe
MOVE sourcedata to StatusRegister if Supervisor
PRIVILEGIZÁLT UTASÍTÁS!

Művelet	**Assembler szintaxis**
MOVE "ea"-t SR-be	MOVE "ea", SR
Operandus hossza:	szó

Kapcsolók:

X	N	Z	V	C
*	*	*	*	*

X a forrásoperandusnak megfelelően állítódik.
N a forrásoperandusnak megfelelően állítódik.
Z a forrásoperandusnak megfelelően állítódik.
V a forrásoperandusnak megfelelően állítódik.
C a forrásoperandusnak megfelelően állítódik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	0	1	1	0	1	1	effektív cím					

effektív cím	a forrásoperandust jelöli; a **adatok** kategória valamennyi címzésmódja megengedett.

Az utasítás leírása:

Az adatok a forráshelyről az állapotregiszterbe kerülnek. Mivel ez az utasítás a supervisor-bitet módosíthatja, ezért ez a privilegizált folyamatok számára van fenntartva.

MOVE from SR ha supervisor, SR –: cél

adatok átvitele az állapotregiszterből a célba
MOVE StatusRegister to destination (if supervisor)
PRIVILEGIZÁLT UTASÍTÁS (68000-esnél nem!)

Művelet	**Assembler szintaxis**
MOVE Sr-t "ea"-ba	MOVE SR, "ea"
Operandus hossza:	szó

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	0	0	0	0	1	1	effektív cím					

effektív cím	a céloperandust jelöli; az **„adatok * változtatható”** kategóriák valamennyi címzésmódja megengedett.

Az utasítás leírása:

Az adatok az állapotregiszterből a célba kerülnek. Ez az utasítás a 68000-esnél nem privilegizált, a 68010-esnél és a 68020-asnál viszont privilegizált.

MOVE USP

ha supervisor, Ax –: USP
vagy ha supervisor, USP –: Ax

a user-veremmutató betöltése vagy elmentése
MOVE to or from UserStackPionter if supervisor
PRIVILEGIZÁLT UTASÍTÁS!

Művelet	**Assembler szintaxis**
MOVE Ax-et USP-be	MOVE Ax, USP
MOVE USP-t Ax-be	MOVE USP, Ax
Operandus hossza:	kettős-szó

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	1	1	1	0	0	1	1	0	dr	Ax reg.		

dr	ez a bit az átvitel irányát (DiRection) adja meg, aminek megfelelően az adatokat mozgatni kell: 0 = Ax –: USP 1 = USP –: Ax
Ax reg.	azt a címregisztert adja meg, ahonnan vagy ahova az adatokat át kell vinni.

Az utasítás leírása:

Mivel a 68000-es – attól függően, hogy melyik üzemmód van bekapcsolva – két különböző veremmutató között tud ide-oda kapcsolni, arra is lehetőséget kell adni, hogy supervisorként a user-veremmutatót inicializálni vagy korrigálni lehessen, mivel a supervisornak általában olyan adminisztratív feladatai vannak, mint egy folyamat inicializálása vagy felügyelete. A supervisor számára ezt a lehetőséget adja meg ez az utasítás. Ezzel az utasítással akár a USP (user-veremmutató) tartalma átmásolható egy címregiszterbe, akár egy címregiszter tartalma beírható a USP-be.

Annak az oka, hogy ez az utasítás miért privilegizált, abban keresendő, hogy a 68000-esnek valószínűleg nincs sem fix USP-je, sem fix SSP-je, hanem mindig két regiszter között válogat. Azért, hogy user-módban a MOVE Ax, USP utasítással ne lehessen az SSP-t felülírni, ez a művelet privilegizált. Ugyanakkor a felhasználó (user) a saját veremmutatóját a MOVEA Ax,A7 utasítással könnyedén módosíthatja.

betöltés címregiszterbe
MOVE to Addressregister

Művelet	**Assembler szintaxis**
MOVEA "ea"-t Ax-be	MOVEA "ea", Ax
Operandus hossza:	szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	hossz		Ax reg.			0	0	1	y módus			reg. y		

hossz	két értéket vehet fel: 11 = szó hosszúságú művelet előjelhelyesen kibővített operandussal 10 = kettős-szó hosszúságú művelet
Ax reg.	a nyolc címregiszter közül annak a számát tartalmazza, amely a cél.
y módus reg. y	ez a két adat az effektív forráscímet határozza meg, valamennyi címzésmód megengedett.

•

Az utasítás leírása:

A MOVEA utasítás egy célként megadandó címregiszterbe betölti az effektív forráscímen levő tartalmat. Az esetleges szó hosszúságú operandus előjelhelyesen kettős-szó formátumra bővül úgy, hogy mindig 32 bit kerül betöltésre.

MOVEC

forrás –: cél

ellenőrző regiszter betöltése vagy elővétele
68010-es és 68020-as processzorok
PRIVILEGIZÁLT UTASÍTÁSA

Művelet	**Assembler szintaxis**
MOVEC Rc-t Rx-be	MOVEC Rc, Rx
MOVEC Rx-et Rc-be	MOVEC Rx, Rc
Operandus hossza:	kettős-szó

Kapcsolók

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	1	1	1	0	0	1	1	1	1	0	1	d

Az utasításszót egy másik szó követi:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	AD	Rx reg.			vezérlőregiszter											

d	ez a bit határozza meg az átvitel irányát: 0 = vezérlőregiszter –: Rx 1 = Rx –: vezérlőregiszter

AD	ez a bit írja le közelebbről az utána következő regisztert: 0 = adatregiszter 1 = címregiszter
Rx reg.	a mindenkori nyolc adat-, ill. címregiszter egyikének a számát tartalmazza.
kontroll-regiszter	ez a 12 bit határozza meg a vezérlőregisztert az alábbi táblázat szerint (a számok hexadecimális értékek): 000 = Source Function Code Register 001 = Destination Function Code Register 002 = CAche Control Register 800 = User Stack Pointer 801 = Vector Base Register 802 = CAche Address Register 803 = Master Stack Pointer 804 = Interrupt Stack Pointer

Az itt fel nem sorolt kódok valamelyikének a megadása egy ILLEGÁLIS UTASÍTÁS feldolgozáshoz vezet.

Az utasítás leírása:

A MOVEC vagy egy megadandó célregiszterbe tölti be az ugyancsak megadandó vezérlőregiszter tartalmát, vagy fordítva.
Ez az utasítás a 68000-es és a 68008-as processzoroknál nem létezik, a többi processzortípusnál pedig privilegizált, mert ezzel a rendszer szempontjából fontos paraméterekhez lehet hozzáférni.

több regiszter töltése vagy tárolása
MOVE Multiple registers

Művelet	**Assembler szintaxis**
MOVEM "reg.lista" "ea"-ba	MOVEM.X "reg.lista", "ea"
MOVEM "ea" "reg. listá"-ba	MOVEM.X "ea", "reg.lista"
Operandus hossza:	szó vagy kettős-szó

Kapcsolók

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15 14 13 12	11	10	9 8 7	6	5 4 3 2 1 0
	0 1 0 0	1	dr	0 0 1	sz	effektív cím

Közvetlenül a műveleti kód után áll a regiszterlista, ami ugyancsak szó hosszúságot foglal el.

dr	az adatátvitel irányát adja meg: 0 = regiszter –: tár 1 = tár –: regiszter
sz	az operandusok hosszát adja meg: 0 = szóoperandus; a tárból egy szónak a regiszterbe való töltésekor a szó előjelhelyesen kibővül, és 32 bites értékként helyezkedik el a célregiszterben. 1 = kettős-szó hosszúságú operandusok

effektív cím	megadja a címzésmódot és azt a címet, amelytől kezdve a regisztertartalmakat el kell helyezni, ill. amely címtől kezdve az ott levő információkat a regiszterekbe kell írni. Ha dr=0, akkor a **vezérlés * változtatható** kategóriák valamennyi címzésmódja és kivételként a -(Ax), tehát a címregiszter közvetett címzése előzetes dekrementálással, ha dr=1, akkor a **vezérlés** kategória, valamint kivételként az (Ax)+, tehát a címregiszter közvetett címzése, utólagos inkrementálással lehetséges.
regiszterlista	azt adja meg, hogy a művelet a CPU mely regisztereit érinti. Ha a maszk megfelelő bitje magasra van állítva, akkor az a regiszter változik, egyébként nem:

reg.: A7 A6 A5 A4 A3 A2 A1 A0 D7 D6 D5 D4 D3 D2 D1 D0
bit: 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Ha a címzésmód a -(Ax), tehát a „normál” veremcímzésmód, akkor a regiszterlista pontosan a fordított:

reg.: D0 D1 D2 D3 D4 D5 D6 D7 A0 A1 A2 A3 A4 A5 A6 A7
bit: 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Az utasítás leírása:

A MOVEM az az utasítás, ami igen kényelmes módon lehetővé teszi majdnem a teljes CPU-állapot tárolását vagy inicializálását. Ehhez csupán egy regiszterlistát kell megadni, amelyben eldöntjük, hogy mely regisztereket kell tárolni, ill. inicializálni, valamint egy effektív címet, amelyet a tár egy indulócímként értelmez, amelytől kezdve a tárolandó adatokat el kell helyezni, ill. amelytől kezdve az inicializáláshoz szükséges értékek vannak. Továbbá meghatározható, hogy a regiszterek töltése szó vagy kettős-szó hosszúságú legyen. Ha szó hosszúságú operandusokról van szó, akkor az adatok a regiszterekbe való beírás előtt előjelhelyesen kettős-szó hosszúságúra bővülnek. Normál esetben a tárolandó adatok a megadott címtől kezdődően, a tárban egymást követő címeken kerülnek elhelyezésre. Ha a címzésmód a veremcímzésmód, tehát -(Ax), azaz címregiszter közvetett címzése előzetes dekrementálással, akkor a regiszterek logikusan nem „alulról felfelé”, hanem fordított sorrendben helyezkednének el. Azért, hogy ezt megakadályozzuk, ennél a címzésmódnál a regiszterlistát egyszerűen meg kell fordítani, hogy a tárban az adatok „helyes” címekre kerüljenek.
Ha az adatátvitel során tárcímeken levő tartalmakat írunk a regiszterekbe, akkor a megadott regiszterekbe azok az értékek töltődnek, amelyek a tárban az „ea”-val megadott címtől kezdődően, egymást követően helyezkednek el.

adatok tárolása vagy olvasása periféria-címről
MOVE data from or to Peripherals

Művelet	**Assembler szintaxis**
MOVEP Dx-et Ay-ba (ind.) + (kül.)	MOVEP Dx,kül.(Ay)
MOVEP Ay(ind.) + (kül.) Dx-be	MOVEP kül.(Ay), Dx
Operandus hossza:	szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	Dx reg.			módus			0	0	1	Ay reg.		

Az utasításszót egy újabb szó követi, ha az Ay címregiszter tartalmához a címkülönbséget tartalmazza.

Dx	annak az adatregiszternek a számát tartalmazza, amelyre vagy amelyről az adatátvitelnek meg kell történnie.
módus	meghatározza az adatátvitel irányát, ill. hosszát, négy értéket vehet fel:

100 = szó átvitele a tárból a regiszterbe
101 = kettős-szó átvitele a tárból a regiszterbe
110 = szó átvitele a regiszterből a tárba
111 = kettős-szó átvitele a regiszterből a tárba

Ay reg.	annak a hivatkozott címregiszternek a számát tartalmazza, amely ennél az utasításnál a **címregiszter közvetett címzése különbséggel** nevű címzésmódnál mindig alkalmazásra kerül.

Az utasítás leírása:

Ez az utasítás szemantikailag egy egyszerű MOVE művelethez hasonlít. Egy adatregiszter és egy tárcím közötti folyamat értéke egy címregiszter és egy konstans különbsége. A „normál" művelettel szemben itt a címkülönbség minden egyes byte után nem eggyel, hanem kettővel növekszik. Ennek az a hatása, hogy az olvasandó vagy írandó szó, ill. kettős-szó nem n egymást közvetlenül követő byte-ba, hanem vagy mindig páros, vagy mindig páratlan címre kerül attól függően, hogy mi a kül. (Ay) értéke. A címregiszter tartalma ennél az utasításnál, hasonlóan a MOVE utasításhoz, nem változik.

Ennek az utasításnak a technikai értelme az, hogy a 68000-es busza egy felső és egy alsó byte-félre van osztva, a busz felső fele a páros címeknek, az alsó fele a páratlan címeknek felel meg.

Ezt a műveletet azért építették be, hogy a 8 bites perifériák csatlakoztatását megkönnyítsék. Ha a perifériaegység pl. a cimbusz felső felét foglalja el, és erre a címre egy szót kell küldeni, akkor az információ egyik fele a periféria alapcímére, a másik fele pedig a busz következő, *ugyanazon felső felére,* tehát két byte-címmel feljebb kerül elhelyezésre.

Egy kettős-szó átvitelénél az információ mindig négy, egymás után következő páros, ill. páratlan címre kerül.

MOVEQ

adatok –: cél

8 bites közvetlen érték gyors betöltése adatregiszterben
MOVE immediate Quick

Művelet	**Assembler szintaxis**
MOVEQ "adat" Dx-be	MOVEQ # "adat", Dx
Operandus hossza:	kettős-szó

Kapcsolók:

X	N	Z	V	C
–	*	*	0	0

X nem változik.
N értéke 1 lesz, ha az eredmény negatív, egyébként törlődik.
Z értéke 1 lesz, ha ez eredmény nulla, egyébként törlődik.
V törlődik.
C törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	1	1	Dx reg.			0	8 bites adatszó							

Dx reg.	annak az adatregiszternek a számát határozza meg, amelybe az adatokat be kell tölteni.
adatszó	ez az a 8 bit szélességű szó, amely előjelhelyesen 32 bit szélességű kettős-szóra bővül, és a megadott adatregiszterbe kerül.

Az utasítás leírása:

Ez az utasítás egy közvetlenül megadandó 8 bites szót egy ugyancsak megadandó adatregiszterbe tölt azt követően, hogy ez a szó előjelhelyesen 32 bitre bővül.

Adatok betöltése vagy olvasása abszolút címről
68010-es és 68020-as processzornál
PRIVILEGIZÁLT UTASÍTÁS

Művelet	**Assembler szintaxis**
MOVES Rx-et DFC-címre	MOVES Rx, "ea"
MOVES SFC-címről Rx-be	MOVES "ea", Rx
Operandus hossza:	byte, szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	1	1	1	0	hossz		effektív cím					

Az utasításszót egy újabb szó követi, ami a regisztert specifikálja:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Ad	Rx reg.			d	0	0	0	0	0	0	0	0	0	0	0

hossz	az operandus hosszától függően három értéket vehet fel: 00 = byte 01 = szó 10 = kettős-szó
effektív cím	a célt vagy a forrást adja meg, a **változtatható** kategória valamennyi címzésmódja megengedett.
AD	az Rx regisztert pontosítja: 0 = adatregiszter 1 = címregiszter
Rx reg.	annak a kiválasztott adat- vagy címregiszternek a számát tartalmazza, amely forrásként vagy célként szolgál.
d	az adatátvitel irányát határozza meg: 0 = "ea" –: Rx 1 = Rx –: "ea"

Az utasítás leírása:

Ez az utasítás a Source Function Register (SFC) vonatkozásában megadott effektív címről byte-ot, szót vagy kettős-szót tölt a megadott cím- vagy adatregiszterbe, illetve a cím- vagy adatregiszterből az értéket a Destination Function Register (DFC) vonatkozásában az effektív cím által megadott tárcímre tölti. Ha egy adatregiszterbe való átvitelről van szó, akkor a betöltendő érték a célregiszter legalsó bitjeire töltődik, ha viszont egy címregiszterbe való átvitelről van szó, akkor az adatok – amennyiben szükséges – előjelhelyesen, 32 bitre kibővítve kerülnek elhelyezésre.

előjelhelyes, bináris szorzás
MULtiply with Sign

Művelet	**Assembler szintaxis**
MULS Dx "ea"-val	MULS "ea", Dx
Operandus hossza:	szó

Kapcsolók:

X	N	Z	V	C
–	*	*	0 *	0

X nem változik.
N értéke 1 lesz, ha az eredméány negatív, egyébként törlődik.
Z értéke 1 lesz, ha az eredmény nulla, egyébként törlődik.
V törlődik, a 68020-asnál a négy szavas struktúra következtében túlcsordulás léphet fel, amit ez a kapcsoló jelez.
C törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	0	0	Dx reg.			1	1	1	effektív cím					

Dx	a vonatkozó adatregiszter számát tartalmazza, amely forrásként és célként van megadva.
effektív cím	a forrásoperandust adja meg; az **adatok** kategória valamenyi címzésmódja megengedett.

Az utasítás leírása:

Az utasítás előjelhelyesen összeszorozza a forrás és a cél tartalmát, mindkettő esetében csak szó hosszúságot fogad el. A 16 bit * 16 bit szorzásnál keletkező 32 bites eredmény a célként megadott adatregiszterbe kerül, és a kapcsolók módosulnak.

MULU cél * forrás –: cél

bináris szorzás az előjel figyelmen kívül hagyásával
MULtiply without Sign

Művelet	**Assembler szintaxis**
MULU Dx "ea"-val	MULU "ea", Dx
Operandus hossza:	szó

Kapcsolók:

X	N	Z	V	C
–	*	*	0 *	0

X nem változik.
N értéke 1 lesz, ha az eredmény legnagyobb helyiértékű bitje magas, egyébként törlődik.
Z értéke 1 lesz, ha az eredmény nulla, egyébként törlődik.
V törlődik, a 68020-asnál a négyszavas struktúra következtében túlcsordulás léphet fel, amit ez a kapcsoló jelez.
C törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	0	0	Dx reg.			0	1	1	effektív cím					

Dx	a vonatkozó adatregiszter számát tartalmazza, amely forrásként vagy célként van megadva.
effektív cím	a forrásoperandust adja meg; az **adatok** kategória valamenynyi címzésmódja megengedett.

Az utasítás leírása:

Az utasítás az előjel figyelmen kívül hagyásával összeszorozza a forrás és a cél tartalmát, mindkettő esetében csak szó hosszúságot fogad el. A 16 bit * 16 bit szorzásánál keletkező 32 bites eredmény a célként megadott adatregiszterbe kerül, és a kapcsolók módosulnak.

N B C D

$0 - \text{cél}_{10} - X$ –: cél

BCD-byte negálása
Negate BCD-Byte

Művelet	Assembler szintaxis
NBCD "ea"	NBCD "ea"
Operandus hossza:	byte

Kapcsolók:

X	N	Z	V	C
*	U	*	U	*

X értéke 1 lesz, ha negatív decimális átvitel lép fel, egyébként törlődik.
N nem határozott.
Z értéke 1 lesz, ha az eredmény nulla, egyébként törlődik.
V nem meghatározott.
C értéke 1 lesz, negatív decimális átvitel lép fel, egyébként törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	1	0	0	0	0	0	effektív cím					

effektív cím	megadja az operandus számára a címzési módot; az "**adatok * változtatható**" kategóriák címzésmódjai használhatók.

Az utasítás leírása:

A megadott célbyte és a X bővítő bit nullából kivonásra kerül, a célnak egy binárisan kódolt decimális számnak kell lennie. A kivonás eredménye ugyancsak a célban kerül elhelyezésre, a kapcsolók megfelelően módosulnak.

negálás
Negate

Művelet	**Assembler szintaxis**
NEG "ea"	NEG.X "ea"
Operandus hossza:	byte, szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
*	*	*	*	*

X értéke 1 lesz, ha negatív decimális átvitel lép fel, egyébként törlődik.
N értéke 1 lesz, ha az eredmény negatív, egyébként törlődik.
Z értéke 1 lesz, ha az eredmény nulla, egyébként törlődik.
V értéke 1 lesz, ha a túlcsordulás keletkezik, egyébként törlődik.
C értéke 1 lesz, ha negatív decimális átvitel lép fel, egyébként törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	0	1	0	0	hossz		effektív cím					

hossz	az operandus hosszát határozza meg, és három értéket vehet fel: 00 = byte 01 = szó 10 = kettős-szó
effektív cím	megadja az operandus számára a címzési módot; az **adatok * változtatható** kategóriák címzésmódjai használhatók.

Az utasítás leírása:

A megadott célbyte nullából kivonásra kerül, a kivonás eredménye ugyancsak a célban kerül elhelyezésre, a kapcsolók megfelelően módosulnak.

negálás a bővítő bit figyelembevételével
NEGate with eXtend

Művelet	**Assembler szintaxis**
NEGX "ea"	NEGX.X "ea"
Operandus hossza:	byte, szó vagy kettős-szó

Kapcsolók

X	N	Z	V	C
*	*	*	*	*

X értéke 1 lesz, ha negatív decimális átvitel lép fel, egyébként törlődik.
N értéke 1 lesz, ha az eredmény negatív, egyébként törlődik.
Z értéke 1 lesz, ha az eredmény nulla, egyébként törlődik.
V értéke 1 lesz, ha túlcsordulás keletkezik, egyébként törlődik.
C értéke 1 lesz, ha negatív decimális átvitel lép fel, egyébként törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	0	0	0	0	hossz		effektív cím					

hossz	meghatározza az operandus hosszát, és három értéket vehet fel: 00 = byte 01 = szó 10 = kettős-szó
effektív cím	megadja az operandus számára a címzési módot, az **adatok * változtatható** kategóriák címzésmódjai használhatók.

Az utasítás leírása:

A megadott célbyte a bővítő bittel együtt nullából kivonásra kerül, a kivonás eredménye ugyancsak a célban kerül elhelyezésre, a kapcsolók megfelelően módosulnak.
Mivel ez a művelet elsőként elvégez egy nullára vonatkozó vizsgálatot, különösen alkalmas arra, hogy hosszabb számok nullával való egyezőségét megvizsgálja.

nincs művelet
No OPeration

Művelet	**Assembler szintaxis**
NOP	NOP
Operandus hossza:	nincs operandus

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	1

Az utasítás leírása:

Ez az utasítás azon kívül, hogy a programszámlálót változtatja, a processzor állapotában semmiféle változást sem idéz elő. Ez az utasítás általában hibakereséshez és időzítésekhez használatos.

logikai NEM
logical NOT

Művelet	**Assembler szintaxis**
NOT "ea"	NOT.X "ea"
Operandus hossza:	byte, szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
–	*	*	0	0

X nem változik.
N értéke 1 lesz, ha az eredmény legnagyobb helyiértékű bitje magas, egyébként törlődik.
Z értéke 1 lesz, ha az eredmény nulla, egyébként törlődik.
V törlődik.
C törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	0	1	1	0	hossz		effektív cím					

hossz	az operandus hosszát határozza meg, és három értéket vehet fel: 00 = byte 01 = szó 10 = kettős-szó
effektív cím	az operandus számára a címzési módot adja meg, az **adatok * változtatható** kategóriák címzésmódja megengedett.

Az utasítás leírása:

A cél tartalma az 1-es komplemensére cserélődik ki.

logikai VAGY
logical OR

Művelet	**Assembler szintaxis**
OR Dx és "ea"	OR.X Dx, "ea"
OR "ea" és Dx	OR.X "ea", Dx
Operandus hossza:	byte, szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
–	*	*	0	0

X nem változik.
N értéke 1 lesz, ha az eredmény legnagyobb helyiértékű bitje magas, egyébként törlődik.
Z értéke 1 lesz, ha az eredmény nulla, egyébként törlődik.
V törlődik.
C törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	0	0	Dx reg.			módus			effektív cím					

Dx reg.	a nyolc adatregiszter egyikének a számát tartalmazza.
módus	meghatározza, hogy az adatregiszter cél vagy forrás, és az operandus hosszát:

byte	**szó**	**kettős-szó**	**irány**
000	001	010	Dx a cél
100	101	110	Dx a forrás

effektív cím	megadja a címzési módot a második operandus számára, az erre vonatkozó értékek a módustól függnek: ha az "ea" a forrás, akkor csak az **adatok** kategória címzésmódjai megengedettek. Ha az "ea" a cél, akkor a **tár * változtatható** kategóriák valamennyi címzésmódja megengedett.

Az utasítás leírása:

A forrás és a cél tartalma között logikai VAGY művelet kerül elvégzésre, az eredmény a célba kerül, és a kapcsolók az eredménynek megfelelően módosulnak.

logikai VAGY közvetlen értékkel
logical OR with immediate value

Művelet	**Assembler szintaxis**
ORI "adat" és "ea"	ORI.X # "adat", "ea"
Operandus hossza:	byte, szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
–	*	*	0	0

X nem változik.
N értéke 1 lesz, ha az eredmény legnagyobb helyiértékű bitje magas, egyébként törlődik.
Z értéke 1 lesz, ha az eredmény nulla, egyébként törlődik.
V törlődik.
C törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	0	hossz		effektív cím					

Az utasításszót egy szó követi, ha az operandus byte vagy szó hosszúságú, és egy kettős-szó követi, ha az operandus kettős-szó hosszúságú. Ha az operandus byte hosszúságú, akkor az ezt tartalmazó szónak csak az alsó 8 bitje (0...7 bit) szignifikáns.

hossz	három értéket vehet fel: 00 = byte hosszúságú művelet 01 = szó hosszúságú művelet 10 = kettős-szó hosszúságú művelet
effektív cím	a céloperandust határozza meg; az **"adatok * változtatható"** kategóriák címzésmódjai, az állapotregiszterrel kapcsolatban a **közvetlen** címzésmódok is alkalmazhatók.

Az utasítás leírása:

Ez az utasítás egy konstans érték és egy megadandó cél közötti logikai VAGY műveletet végez el, és beállítja a megfelelő kapcsolókat. A konstans érték a tárban közvetlenül az utasításszó után helyezkedik el, és lehet byte, szó vagy kettős-szó hosszúságú. A byte hosszúságú operandusok szó hosszúságú címeket foglalnak el.

Ha célként az állapotregiszterről van szó, akkor ehhez kétféle hozzáférési lehetőség van:

Ha byte hosszúságú műveletről van szó, akkor az állapotregiszternek csak a USER fele kerül felhasználásra.

Ha szó hosszúságú műveletről van szó, akkor ez az utasítás egy PRIVILEGIZÁLT UTASÍTÁS.

P A C K

$\text{forrás}_{\text{normál BCD}}$ + offszet – : $\text{cél}_{\text{tömörített BCD}}$

két BCD-szám tömörítése
PACK BCD
CSAK A 68020-AS PROCESSZORNÁL

Művelet	**Assembler szintaxis**
PACK Dx, Dy offszettel	PACK Dx, Dy, # "adat"
PACK -(Ax), -(Ay) offszettel	PACK -(Ax), -(Ay), # "adat"
Operandus hossza:	szó

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	0	0	Ry reg.			1	0	1	0	0	RM	Rx reg.		

Az utasítás-szót egy újabb szó követ: az offszet.

Ry reg.	a célregisztert specifikálja, amely lehet akár adat-, akár címregiszter.
RM	közelebbről meghatározza a forrást és a célt: 0 = az adatátvitel két adatregiszter között történik 1 = az adatátvitel a táron keresztül, két címregiszterrel történik, a címzésmód csak közvetett lehet, előzetes dekrementálással.

Rx reg. ez a mező a forrásregiszter regiszterszámát tartalmazza függetlenül attól, hogy az adat- vagy címregiszter.

Az utasítás leírása:

Ez az utasítás arra alkalmas, hogy kettő darab, két byte-ra felosztott BCD-számot egymással összekapcsoljon, és azokat egyetlen egy célbyte-ba helyezze el. Ehhez forrás- és célregiszterként vagy két adatregiszter adható meg, vagy pedig két címregiszteren keresztül egy-egy tárcímhez lehet fordulni, amikoris a címregiszterek értelemszerűen előzetes dekrementálásra kerülnek.
A két BCD-számhoz tovább hozzáadható még egy offszet, hogy egy kódkonvertálást lehetőség szerint a tömörítéssel egyidőben el lehessen végezni.
Ha célként és forrásként adatregiszterek vannak megadva, akkor a tömörítendő számok a forrás alsó ill. felső szavában vannak. Ezután ehhez a forrásregiszterhez hozzáadódik az offszet, és az eredmény 11...8 bitjei a célregiszter 7...4 bitjeibe, az eredmény 3...0 bitjei pedig a célregiszter 3...0 bitjeibe másolódnak át.
A célregiszter magasabb helyiértékű szava nem változik.
Ha az adatok feldolgozása nem adatregiszterek között, hanem közvetett módon, címregisztereken keresztül történik, akkor ez némiképp másként megy végbe:
Először a forráscímen levő két, egymást követő szó egy kettős-szóban kerül egymás mellé (a tárban a nagyobb címen levő szó lesz a magasabb helyiértékű szó), és az offset ehhez a kettős-szóhoz adódik hozzá. Ebből az eredményből a 11...8 bitek a célregiszter 7...4 bitjeibe, a 3...0 bitek a célregiszter 3...0 bitjeibe másolódnak át.

effektív cím tárolása a verembe
Push Effektiv Address

Művelet	**Assembler szintaxis**
PEA ea	PEA ea
Operandus hossza:	kettős-szó

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	1	0	0	0	0	1	effektív cím					

effektív cím	a céloperandust határozza meg; a **vezérlés** kategória valamennyi címzésmódja megengedett.

Az utasítás leírása:

Az effektív cím, kettős-szó hosszúságban, a szokásos módon a verembe kerül.

RESET vezeték aktiválása
PRIVILEGIZÁLT MŰVELET

Művelet	**Assembler szintaxis**
RESET	RESET
Operandus hossza:	nincs operandus

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	0

Az utasítás leírása:

A 68000-es RESET-vezetéke 124 ütemciklus időtartamon keresztül aktiválódik. Ez lehetőséget ad a processzornak arra, hogy ha a periféria egységeknél valamilyen hibát észlel, akkor ezeket adott körülmények között egy RESET impulzussal meghatározott állapotba vigye, hogy azokat szoftver útján újra lehessen inicializálni.
Ez az utasítás a processzor állapotát csak annyiban változtatja meg, mint a NOP utasítás.
Ez a művelet, tekintettel arra, hogy a rendszerre végzetes hatást gyakorolhat, privilegizált művelet.

ROL/ROR

cél eltolva – : cél

egy regiszter bitenkénti elforgatása balra/jobbra
ROtate Left/Right

Művelet	**Assembler szintaxis**
ROx Dx-et Dy hellyel	ROx.X Dy, Dx
ROx Dx-et "adat" hellyel	ROx.X #"adat", Dx
ROx "ea"-t 1 hellyel	ROx.X "ea"
Operandus hossza:	byte, szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
–	*	*	0	*

X nem változik.
N értéke 1 lesz, ha az eredmény legnagyobb helyiértékű bitje magas, egyébként törlődik.
Z értéke 1 lesz, ha az eredmény nulla, egyébként törlődik.
V törlődik.
C megegyezik az operandusból legutoljára „kiforgatott" bittel, ill. törlődik, ha a forgatási helyek száma nulla.

Az utasítás formátuma egy adatregiszter elforgatásához:

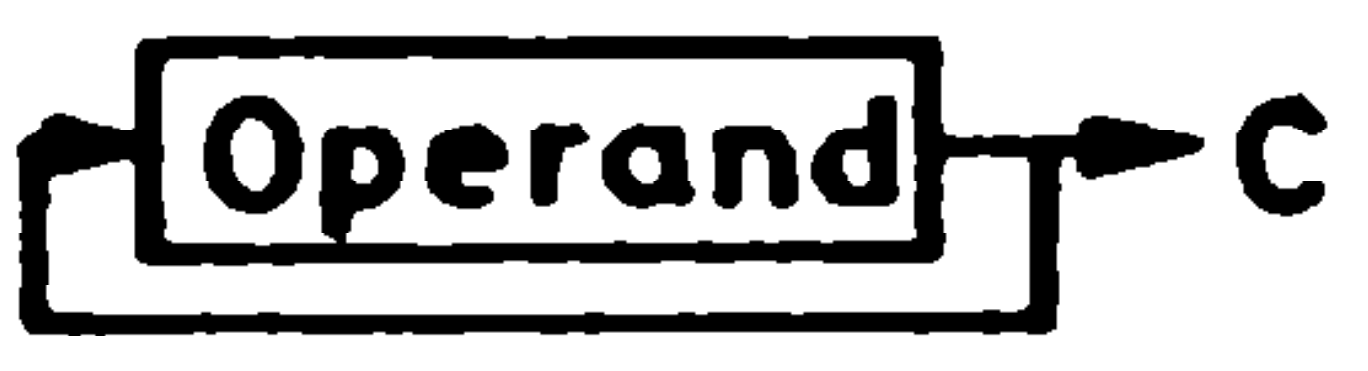
bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	0	Dy reg.			dr	hossz		i	1	1	Dx reg.		

Dy reg.	a nyolc adatregiszter (Dy) közül annak a számát adja meg, amelyik meghatározza eltolások számát, ill. meghatározza az abszolút értéket ("adat"), az i bittől függően. Egy esetleges abszolútérték szám szerint 0 és 7 között (3 bit) lehet, ahol a 0 érték 8 hellyel való eltolásnak felel meg.
dr	ez a bit az elforgatás irányát (DiRection) adja meg: 0 = elforgatás jobbra 1 = elforgatás balra

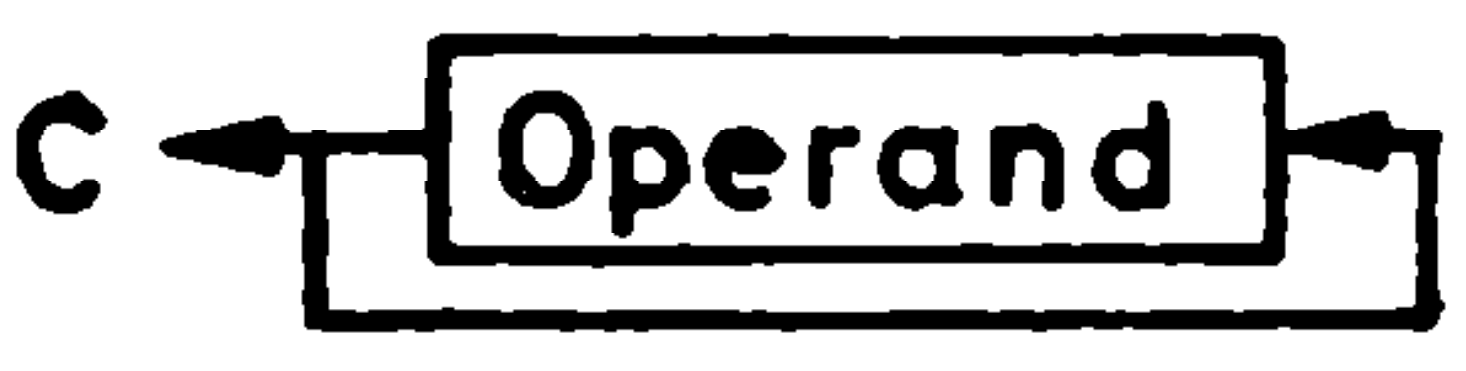
hossz	három értéket vehet fel: 00 = byte hosszúságú művelet 01 = szó hosszúságú művelet 10 = kettős-szó hosszúságú művelet
i	azt határozza meg, hogy a Dy reg. egy regisztert vagy egy abszolút értéket jelöl: 0 = abszolút érték 1 = Dy reg. egy adatregisztert jelöl
Dx reg.	ez a három bit azt az adatregisztert specifikálja, amelynek a tartalmát el kell forgatni (a cél címe).

Az utasítás leírása:

Ez az utasítás egy megadandó cél tartalmának egy ugyancsak megadandó számú bináris hellyel való jobbra vagy balra forgatását végzi.
Az elforgatás irányát az assembler különböző szintaxissal jelöli (ROL/ROR), de a kódok a két esetben csak egy bitben különböznek egymástól.
A cél címként egy adatregiszter adható meg. Ebben az esetben a cél címe (tehát az adatregiszter) mellett még az elforgatások számát is meg kell adni. Ezt meg lehet adni egy konstans értékként, vagy egy másik adatregiszterben. Ha viszont célként nincs adatregiszter specifikálva, akkor a **tár * változtatható** kategóriák valamennyi címzésmódja megengedett, de az elforgatások száma nem; ez ebben az esetben mindig 1.
Továbbá figyelembe kell venni, hogy az operandus hossza (byte, szó vagy kettős-szó) csak regisztereltolás esetén választható meg. A tárban levő operandus esetén az operandus hossza mindig szó.



R O R



R O L

Az utasítás formátuma ha az elforgatás a tárban történik:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	0	0	1	1	dr	1	1	effektív cím					

dr	ez a bit az elforgatás irányát (DiRection) adja meg: 0 = elforgatás jobbra 1 = elforgatás balra
effektív cím:	a céloperandust határozza meg; az **adatok * változtatható** kategóriák valamennyi címzésmódja megengedett.

egy regiszter bitenkénti elforgatása balra/jobbra bővítő bittel
ROtate Left/Right with eXtendbit

Művelet	**Assembler szintaxis**
ROXx Dx-et Dy hellyel	ROx.X Dy, Dx
ROXx Dx-et ”adat” hellyel	ROx.X #”adat”, Dx
ROXx ”ea”-t 1 hellyel	ROx.X ”ea”
Operandus hossza:	byte, szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
*	*	*	0	*

X megegyezik az operandusból legutoljára „kiforgatott” bittel, ill. nem változik, ha a helyek száma nulla.
N értéke 1 lesz, ha az eredmény legnagyobb helyiértékű bitje magas, egyébként törlődik.
Z értéke 1 lesz, ha az eredmény nulla, egyébként törlődik.
V törlődik.
C megegyezik az operandusból legutoljára „kiforgatott” bittel, ill. az X értékével, ha a forgatási helyek száma nulla.

Az utasítás formátuma egy adatregiszter elforgatásához:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	0	Dy reg.			dr	hossz		i	1	0	Dx reg.		

Dy reg.	a nyolc adatregiszter (Dy) közül annak a számát adja meg, amelyik meghatározza eltolások számát, ill. az abszolút értéket (”adat”), az i bittől függően. Egy esetleges abszolútérték szám szerint 0 és 7 között (3 bit) lehet, ahol a 0 érték 8 hellyel való eltolásnak felel meg.
dr	ez a bit az elforgatás irányát (DiRection) adja meg: 0 = elforgatás jobbra 1 = elforgatás balra
hossz	három értéket vehet fel: 00 = byte hosszúságú művelet 01 = szó hosszúságú művelet 10 = kettős-szó hosszúságú művelet

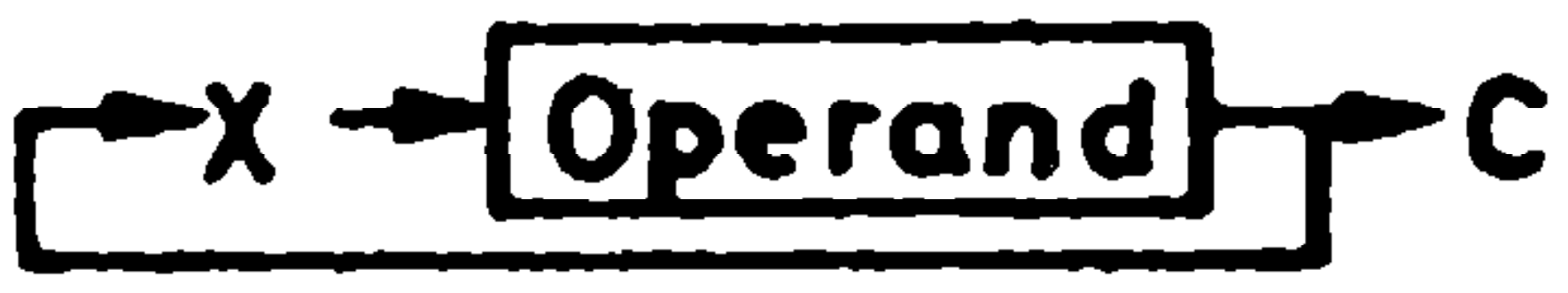
i — azt határozza meg, hogy a Dy reg. egy regisztert vagy egy abszolút értéket jelöl:

0 = abszolút érték
1 = Dy reg. egy adatregisztert jelöl

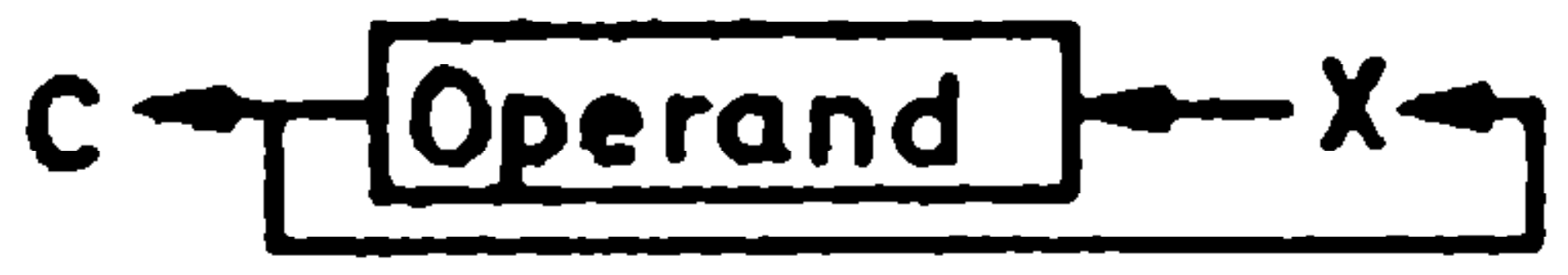
Dx reg. — ez a három bit azt az adatregisztert specifikálja, amelynek a tartalmát el kell forgatni (a cél címe).

Az utasítás leírása:

Ez az utasítás egy megadandó cél tartalmának egy ugyancsak megadandó számú bináris hellyel való jobbra vagy balra forgatását végzi, amelynek során az elforgatott tartalom áthalad az X biten.
Az elforgatás irányát az assembler különböző szintaxissal jelöli (ROL/ROR), de a kódok a két esetben csak egy bitben különböznek egymástól.
A cél címként egy adatregiszter adható meg. Ebben az esetben a cél címe (tehát az adatregiszter) mellett még az elforgatások számát is meg kell adni. Ezt meg lehet adni egy konstans értékként, vagy egy másik adatregiszterben. Ha viszont célként nincs adatregiszter specifikálva, akkor a **tár * változtatható** kategóriák valamennyi címzésmódja megengedett, de az elforgatások száma nem; ez ebben az esetben mindig 1.
Továbbá figyelembe kell venni, hogy az operandus hossza (byte, szó vagy kettős-szó) csak regisztereltolás esetén választható meg. A tárban levő operandus esetén az operandus hossza mindig szó.



ROXR



ROXL

Az utasítás formátuma ha az elforgatás a tárban történik:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	0	0	1	0	dr	1	1	effektív cím					

dr — ez a bit az elforgatás irányát (DiRection) adja meg:

0 = elforgatás jobbra
1 = elforgatás balra

effektív cím: — a céloperandust határozza meg; az **adatok * változtatható** kategóriák valamennyi címzésmódja megengedett.

R T D

(SP) + – : PC;
SP + 4 + param. – : SP

visszatérés alprogramból és a helyi paraméterek törlése
ReTurn from procedure and Deallocate Parameters
68010-es és 68020-as PROCESSZOROK

Művelet	**Assembler szintaxis**
RTD veremkorrekció	RTD # "adat"
Operandus hossza:	nincs operandus

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	0

Az utasítás leírása:

Ez az utasítás befejezi egy műveletsor feldolgozását, miközben a veremből a visszatérési címet visszatölti a programszámlálóba. Ezt követően a veremmutatót áthelyezi arra az esetre, ha a veremben a visszatérési cím után még paraméterek állnának.

visszatérés kizárásból
ReTurn from Exception
PRIVILEGIZÁLT MŰVELET

Művelet	**Assembler szintaxis**
RTE	RTE
Operandus hossza:	nincs operandus

Kapcsolók:

X	N	Z	V	C
*	*	*	*	*

X a verem tartalmának megfelelően változik.
N a verem tartalmának megfelelően változik.
Z a verem tartalmának megfelelően változik.
V a verem tartalmának megfelelően változik.
C a verem tartalmának megfelelően változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1

Az utasítás leírása:

Ez az utasítás egy kizárási feltétel feldolgozását fejezi be, miközben a veremből az állapotregisztert és a visszatérési címet a megfelelő regiszterekbe visszatölti.
Mivel az állapotregiszter betöltésekor a supervisor kapcsoló magasra állítódhat, ez a művelet csak privilegizált processzorok számára van fenntartva. Ezzel adekvát művelet lehet a user-folyamatokhoz az RTR.

tárolt modulállapot újbóli visszatöltése
ReTurn from Module
CSAK 68020-AS PROCESSZORNÁL

Művelet	**Assembler szintaxis**
RTM moduladatmutató	RTM Rx
Operandus hossza:	nincs operandus

Kapcsolók:

X	N	Z	V	C
*	*	*	*	*

X a verem tartalmának megfelelően változik.
N a verem tartalmának megfelelően változik.
Z a verem tartalmának megfelelően változik.
V a verem tartalmának megfelelően változik.
C a verem tartalmának megfelelően változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	1	1	0	1	1	0	0	D	Rx reg.		

D	ez a bit az alkalmazott regiszter típusát jelöli: 0 = az Rx adatregiszter 1 = az Rx címregiszter
Rx reg.	megadja a regiszter számát. Ha moduladat-mutatóként a veremmutató (A7) kerül alkalmazásra, akkor figyelembe kell venni, hogy a modulállapot megismétlése után a mutató korábbi értéke természetesen már nem áll rendelkezésre.

Az utasítás leírása:

Ezzel a művelettel egy korábban egyszer tárolt modulállapotot a veremből visszatölthetünk.
Moduladat-mutatóként értelmezhető egy adat- vagy egy címregiszter, és a modul újbóli betöltése után a verem a modulblokkban megadott paraméterek számának megfelelően korrigálásra kerül.

R T R (SP)+ – : CC; (SP)+ – : PC

visszatérés alprogramból és kapcsolóbitek betöltése a veremből
ReTurn and Restore Condition Codes

Művelet	**Assembler szintaxis**
RTR	RTR
Operandus hossza:	nincs operandus

Kapcsolók:

X	N	Z	V	C
*	*	*	*	*

X a verem tartalmának megfelelően változik.
N a verem tartalmának megfelelően változik.
Z a verem tartalmának megfelelően változik.
V a verem tartalmának megfelelően változik.
C a verem tartalmának megfelelően változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	1

Az utasítás leírása:

A vermből újra betöltődik az állapotregiszter és a programszámláló.
Ez a művelet olyan, mint a Return from Exception művelet, azzal a különbséggel, hogy a veremből csak a feltételkódok kerülnek ismételten betöltésre anélkül, hogy az állapotszó supervisor-része megváltozna. Ez a felhasználó számára lehetőséget ad arra, hogy szoftveroldali megszakításokat vagy kizárásokat vizsgáljon anélkül, hogy supervisor állapotba kellene kerülnie.

R T S (SP)+ – : PC

visszatérés szubrutinból
ReTurn from Subrutine

Művelet	**Assembler szintaxis**
RTS	RTS
Operandus hossza:	nincs operandus

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	1

Az utasítás leírása:

Ez a művelet befejezi egy folyamat feldolgozását, miközben a visszatérési címet a veremből visszatölti a programszámlálóba.

BCD számok kivonása bővítő bittel
Subtract Binary Coded Decimals with extend

Művelet	**Assembler szintaxis**
SBCD Dx-et Dy-ból	SBCD Dx, Dy
SBCD -(ind.) Ax-et -(ind.) Ay-ból	SBCD -(Ax), -(Ay)
Operandus hossza:	byte

Kapcsolók:

X	N	Z	V	C
*	U	*	U	*

X értéke 1 lesz, ha egy negatív decimális átvitel keletkezik, egyébként törlődik.
N nem meghatározható.
Z értéke 1 lesz, ha az eredmény nulla, egyébként nem változik.
V nem meghatározható.
C értéke 1 lesz, ha egy negatív decimális átvitel keletkezik, egyébként törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	0	0	y reg.			1	0	0	0	0	T	x reg.		

y reg.	a célként megadandó cím- vagy adatregiszter számát tartalmazza.
T	a cél- és a forráscímet specifikálja. Ha T=0, akkor az x regiszter és az y regiszter adatregiszterek, egyébként címregiszterek.
x reg.	a forrásként megadandó cím- vagy adatregiszter számát tartalmazza.

Az utasítás leírása:

A cél tartalmából kivonásra kerül a forrás tartalma és a bővítő bit, az eredmény a célba kerül, és a kapcsolók megfelelően állítódnak. Kétféle címzési mód áll rendelkezésre:

- adatregiszter és adatregiszter között: az utasításszóban megadott regiszterek tartalmazzák a forrás- és a célinformációt.
- Közvetett módon, címregisztereken keresztül, előzetes dekrementálással: az operandusokat azokról a tárcímekről vesszük elő, amelyekre az utasításszóban megadott címregiszterek 1-gyel való csökkentés után mutatnak.

S cc

ha cc igaz, akkor #$ff – : cél
ha nem, akkor #$00 – : cél

egy byte magasra állítása a feltételkódoktól függően
Set byte according to Condition-Codes

Művelet	**Assembler szintaxis**
S_{CC} "ea"	S_{CC} "ea"
Operandus hossza:	byte

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	1	feltétel				1	1	effektív cím					

feltétel: itt a következőkben leírt feltételbit-kombinációk egyike áll.

effektív cím: a célbyte-ot adja meg; az **adatok * változtatható** kategóriák valamennyi címzésmódja megengedett.

Az utasítás leírása:

Az utasítás végrehajtásának megkezdésekor felülvizsgálatra kerül az állapotregiszter a megadott feltételkód ($_{CC}$) vonatkozásában. Ha ez teljesül, tehát logikailag igaz, akkor a célbyte-ba a #$FF kerül, ha nem igaz, akkor #$00. Feltételként ($_{CC}$) a következő 14 bitkombináció egyike adható meg:

Név	**Kód**	**Jelentés**	**Név**	**Kód**	**Jelentés**
CC	0100	Carry = 0	LS	0011	nem nagyobb
CS	0101	Carry = 1	LT	1101	kisebb
EQ	0111	egyenlő	MI	1011	negatív
GE	1100	nagyobb/egyenlő	NE	0110	nem egyenlő
GT	1110	nagyobb mint 0	PL	1010	pozitív
HI	0010	nagyobb	VC	1000	túlcsordulás = 0
LE	1111	kisebb/egyenlő	VS	1001	túlcsordulás = 1

E feltételek pontosabb leírását, különösképpen a kapcsolóbitekkel való ábrázolásukat illetően e fejezet elejére utalunk.

állj on le a processzor, miután az állapotregiszter betöltésre került
STOP with conditioncode loaded

Művelet	**Assembler szintaxis**
STOP # "adat"-tal az SR-ben	STOP # "adat"
Operandus hossza:	byte, szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
*	*	*	*	*

X az utasításban levő fix értéknek megfelelően töltődik.
N az utasításban levő fix értéknek megfelelően töltődik.
Z az utasításban levő fix értéknek megfelelően töltődik.
V az utasításban levő fix értéknek megfelelően töltődik.
C az utasításban levő fix értéknek megfelelően töltődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	0

Közvetlenül az utasításszó után van egy újabb szó, amelyik azt a kódot tartalmazza, amit a processzor megállítása előtt az állapotregiszterbe kell tölteni.

Az utasítás leírása:

Ez az utasítás elsősorban arra alkalmas, hogy az állapotregiszterbe egy olyan konstans érték kerüljön, amelyik a műveleti kódot követő szóban van.
Ezt követően a processzor az álljállapotba kapcsol át, tehát aktiválja az álljvezetéket, és addig marad ebben az állapotban, amíg egy olyan, feldolgozandó megszakítás nem lép fel, ami a processzort a RESET-vezetéken keresztül vissza nem állítja az eredeti állapotába, vagy amíg egy kizárás nem következik be. Ez előfordulhat például akkor, ha a STOP utasítás végrehajtásának pillana-

tában a TRACE-bit magas volt, vagy ha az állapotregiszterben levő állandó értéknek a SUPERVISOR kapcsolóbitet vissza kellett állítania. Az első esetben a szoftveroldali megszakítás (TRACE-Trap), a második esetben a „privilégium megsértése" megszakítás kerül behívásra.

Egy megszakítás csak akkor tudja a STOP-állapotot megszüntetni, ha ennek nagyobb a prioritása, mint amit az állapotregiszter éppen jelzett, vagy ha az egy nem maszkolható megszakítás (NMI).

binthe áris kivonás
SUB Binary

Művelet	**Assembler szintaxis**
SUB Dx-et "ea"-ból	SUB.X Dx, "ea"
SUB "ea"-t Dx-ből	SUB.X "ea", Dx
Operandus hossza:	byte, szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
*	*	*	*	*

X értéke 1 lesz, ha egy negatív átvitel keletkezik, egyébként törlődik.
N értéke 1 lesz, ha az eredmény negatív, egyébként törlődik.
Z értéke 1 lesz, ha az eredmény zérus, egyébként törlődik.
V értéke 1 lesz, ha egy negatív túlcsordulás lép fel, egyébként törlődik.
C értéke 1 lesz, ha átvitel keletkezik, egyébként törlődik.

Az utasítás formátuma:

bit	15 14 13 12	11 10 9	8 7 6	5 4 3 2 1 0
	1 0 0 1	reg.	módus	effektív cím

reg. a nyolc adatregiszter egyikének a számát tartalmazza.

módus meghatározza, hogy az adatregiszter a cél vagy a forrás, valamint az operandus hosszát:

byte	**szó**	**kettős-szó**	**irány**
000	001	010	Dx a cél
100	101	110	Dx a forrás

effektív cím a második operandus számára adja meg a címzési módot, az erre vonatkozó értékek a módustól függenek:
Ha az "ea" a forrás, akkor egy kivételével valamennyi címzési mód megengedett: ha az operandus byte hosszúságú,

akkor az An (címregiszter közvetlen címzése) nem megengedett.
Ha az "ea" a cél, akkor a **tár * változtatható** kategóriák valamennyi címzésmódja megengedett.

Az utasítás leírása:

Az utasítás kivonja a forrás tartalmát a cél tartalmából, az eredmény a célba kerül, és a kapcsolóbitek az eredménynek megfelelően változnak.

binàris kivonás a címregiszterből
SUBtract binary from Addressregister

Művelet	**Assembler szintaxis**
SUBA "ea"-t An-ből	SUBA.X "ea", An
Operandus hossza:	szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	0	1	reg.			módus			effektív cím					

reg.	a nyolc adatregiszter közül annak a számát tartalmazza, amelyik a célregiszter.
módus	az operandus hosszát határozza meg: 011 = szó hosszúságú operandus. Azért, hogy a címek kiszámítását helyesen el lehessen végezni, a forrásoperandus előjelhelyesen 32 bitre bővül. 111 = kettős-szó hosszúságú operandus.
effektív cím	a forrásoperandust jelöli; valamennyi címzési mód megengedett.

Az utasítás leírása:

Az utasítás a forrás tartalmát kivonja a célként használt címregiszter tartalmából.
A SUBA felépítését tekintve a SUB utasítástól csak a módust illetően különbözik. Amíg a SUB utasításnál az x00, x01 és x10 értékek kerülnek alkalmazásra, addig a SUBA utasításnál az x11 értékek. Így a SUBA utasítást a SUB utasításnak nem csak szemantikailag különböző címzésmódjának lehet tekinteni.

SUBI

cél – adatok – : cél

konstans érték kivonása
SUBtract Immediate

Művelet	**Assembler szintaxis**
SUBI "adat" "ea"-ból	SUBI.X # "adat", "ea"
Operandus hossza:	byte, szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
*	*	*	*	*

X értéke 1 lesz, ha egy negatív átvitel keletkezik, egyébként törlődik.
N értéke 1 lesz, ha az eredmény negatív, egyébként törlődik.
Z értéke 1 lesz, ha az eredmény nulla, egyébként törlődik.
V értéke 1 lesz, ha túlcsordulás következik be, egyébként törlődik.
C értéke 1 lesz, ha egy negatív átvitel keletkezik, egyébként törlődik.

Az utasítás formátuma

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	1	0	0	hossz		effektív cím					

Az utasítás-szót egy szó követi, ha az operandus hossza byte vagy szó; egy kettős-szó követi, ha az operandus hossza kettős-szó. Ha az operandus hossza byte, akkor a szónak csak az alsó 8 bitje (0...7 bit) szignifikáns.

hossz	három értéket vehet fel: 00 = byte hosszúságú művelet 01 = szó hosszóságú művelet 10 = kettős-szó hoszúságú művelet
effektív cím	a céloperandust határozza meg; csak az **adatok + változtatható** kategóriák címzésmódjai megengedettek.

Az utasítás leírása:

Ez az utasítás egy állandó értéket von ki binárisan egy megadandó célból, és a vonatkozó kapcsolóbiteket módosítja. Az állandó érték a tárban közvetlenül az utasításszó után van, és lehet byte, szó vagy kettős-szó hosszúságú. A byte hosszúságú operandusok szó hosszúságú címeket foglalnak el.

közvetlen, 3 bites szó gyors kivonása
SUBtract immediate Quick

Művelet	**Assembler szintaxis**
SUBQ "adat"-t "ea"-ból	SUBQ.X # "adat", "ea"
Operandus hossza:	byte, szó, kettős-szó

Kapcsolók:

X	N	Z	V	C
*	*	*	*	*

X értéke 1 lesz, ha egy negatív átvitel keletkezik, egyébként törlődik.
N értéke 1 lesz, ha az eredmény negatív, egyébként törlődik.
Z értéke 1 lesz, ha az eredmény nulla, egyébként törlődik.
V értéke 1 lesz, ha túlcsordulás lép fel, egyébként törlődik.
C értéke 1 lesz, ha egy negatív átvitel keletkezik, egyébként törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	1	adatok			1	hossz		effektív cím					

adatok	3 bit szélességű érték, amit az "ea"-ból ki kell vonni.
hossz	három értéket vehet fel: 00 = byte hosszúságú művelet 01 = szó hosszúságú művelet 01 = kettős-szó hosszúságú művelet
effektív cím	a céloperandust határozza meg; csak a **változtatható** kategória címzésmódjai megengedettek. Byte hosszúságú műveletnél továbbá még az An címzésmód (címregiszter közvetlen) is tiltott.

Az utasítás leírása:

Ez az utasítás egy konstans értéket binárisan kivon egy megadandó célból, és módosítja a kapcsolóbiteket. Az állandó érték, ami 0 és 7 közötti értékeket vehet fel (3 bit!), közvetlenül az utasításszóban van, és kivonható byte-ból, szóból vagy kettős-szóból.

bináris kivonás a bővítő bittel
SUBtract binary with eXtend

Művelet	**Assembler szintaxis**
SUBX Dx-et Dy-ból	SUBX.X Dx, Dy
SUBX -(ind.) Ax-et -(ind.) Ay-ból	SUBX.X (-Ax), -(Ay)
Operandus hossza:	byte, szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
*	*	*	*	*

X értéke 1 lesz, egy egy negatív átvitel keletkezik, egyébként törlődik.
N értéke 1 lesz, ha az eredmény negatív, egyébként törlődik.
Z értéke 1 lesz, ha az eredmény nulla, egyébként törlődik.
V értéke 1 lesz, ha túlcsordulás lép fel, egyébként törlődik.
C értéke 1 lesz, ha egy negatív átvitel keletkezik, egyébként törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	0	1	y reg.			1	hossz		0	0	T	x reg.		

y reg.	a célként megadott cím, ill. adatregiszter számát tartalmazza.
hossz	három értéket vehet fel:

00 = byte hosszúságú művelet
01 = szó hosszúságú művelet
10 = kettős-szó hosszúságú művelet

x reg.	a forrásként megadott cím-, ill. adatregiszter számát tartalmazza.
T	a cél- és a forráscímet specifikálja. Ha T=0, akkor az x regiszter és az y regiszter adatregiszterek, egyébként címregiszterek.

Az utasítás leírása:

A cél tartalmából kivonásra kerül a forrás tartalma és a bővítő bit, az eredmény a célba kerül, és a kapcsolók megfelelően állítódnak. Kétféle címzési mód áll rendelkezésre:

- adatregiszter és adatregiszter között: az utasításszóban megadott regiszterek tartalmazzák a forrás- és a célinformációt.
- Közvetett módon, címregisztereken keresztül, előzetes dekrementálással: az operandusokat azokról a tárcímekről vesszük elő, amelyekre az utasításszóban megadott címregiszterek 1-gyel való csökkentés után mutatnak.

a regiszter magasabb és alacsonyabb helyiértékű szavának felcserélése
SWAP register halves

Művelet	**Assembler szintaxis**
SWAP Dx	SWAP Dx
Operandus hossza:	szó

Kapcsolók:

X	N	Z	V	C
–	*	*	0	0

X nem változik.
N értéke 1 lesz, ha az eredmény kettős-szó negatív, egyébként törlődik.
Z értéke 1 lesz, ha az eredmény nulla, egyébként törlődik.
V törlődik.
C törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	1	0	0	0	0	1	0	0	0	Dx reg.		

Dx reg.	annak az adatregiszternek a számát tartalmazza, amelynek a két szavát fel kell cserélni.

Az utasítás leírása:

A megadandó adatregiszter alsó és felső felét egymással kicseréli, és a kapcsolókat az eredménynek megfelelően beállítja.
Mindig egy regiszter felső és alsó szavának cseréje történik meg.

TAS teszt(cél) – : CC; #1 – : $cél_{7.bit}$

egy byte vizsgálata, és legmagasabb bitjének magasra állítása
Test byte And Set always bit 7

Művelet	**Assembler szintaxis**
TAS "ea"	TAS "ea"
Operandus hossza:	byte

Kapcsolók:

X	N	Z	V	C
–	*	*	0	0

X nem változik.
N értéke 1 lesz, ha a vizsgálandó byte negatív volt, egyébként törlődik.
Z értéke 1 lesz, ha a vizsgálandó byte nulla volt, egyébként törlődik.
V törlődik.
C törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	1	0	1	0	1	1	effektív cím					

effektív cím	a vizsgálandó, ill. magasra állítandó byte címét tartalmazza. Az **adatok * változtatható** kategóriák valamennyi címzésmódja megengedett.

Az utasítás leírása:

Ez az utasítás két részműveletből áll: az első részművelet a megadandó byte-ot megvizsgálja az előjelre, ill. a nullával való egyezőségre vonatkozóan, a második részművelet pedig ugyanennek a byte-nak a legmagasabb helyiértékű bitjét 1-re állítja.
Mivel a vizsgálat és a bit állítása közötti időben a busz más processzorok részére nem adható át, ez az utasítás kifejezetten a szemaforok kezelésére alkalmas (lásd a 7. fejezetet).

TRAP

PC – : -(SSP); SR – : -(SSP);
($\text{vektor}_{\text{szám}}$) – : PC

szoftveroldali megszakítás

Művelet	**Assembler szintaxis**
TRAP "vektor"	TRAP # "vektor"
Operandus hossza:	nincs operandus

Kapcsolók

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	1	1	1	0	0	1	0	0	vektor			

vektor — itt annak a felhasználói (user) vektornak a száma áll, amelyet végre kell hajtani. A vektorszámokat, az effektív CPU-vektorokhoz való hozzárendelésüket és a címeket a következő táblázat tartalmazza:

vetor	eff. vektor	cím
\$00	\$20	\$0080
\$01	\$21	\$0084
\$02	\$22	\$0088
\$03	\$23	\$008C
\$04	\$24	\$0090

$05	$25	$0094
$06	$26	$0098
$07	$27	$009C
$08	$28	$00A0
$09	$29	$00A4
$0A	$2A	$00A8
$0B	$2B	$00AC
$0C	$2C	$00B0
$0D	$2D	$00B4
$0E	$2E	$00B8
$0F	$2F	$00BF

Az utasítás leírása:

A processzor ezen utasítás kiadásakor megkezdi a megadott számú szoftveroldali megszakítás (trap) utasítássorozatának a feldolgozását. Ez azzal kezdődik, hogy az állapotregiszter és a programszámláló pillanatnyi tartalmát tárolja a supervisor-verembe. Ezután a megszakítás számának megfelelő vektorcímen levő kettős-szót mint új programszámlálót betölti, és a további feldolgozást ettől a címtől kezdődően folytatja.

TRAP$_{cc}$

CC: PC –: -(SSP); SR –: -(SSP);
($\text{vektor}_{\text{szám}}$ 7) –: PC

szoftveroldali megszakítás, ha a feltételek teljesülnek
TRAP on Condition-Codes true
CSAK A 68020-AS PROCESSZORNÁL

Művelet	**Assembler szintaxis**
TRAP$_{CC}$ paraméterrel	TRAPcc.X # "adat"
Operandus hossza:	szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	1	feltétel				1	1	1	1	1	módus		

Az utasításszó után vagy nem áll szó, vagy következi egy vagy két újabb szó, amelyek szabadon használhatók a megszakításhoz.

feltétel:	itt a következőkben leírt feltételbit-kombinációk valamelyike áll.
módus	megállapítja, hogy az utasításszó után állnak-e paraméterek, ill. ha igen, akkor milyen hosszúak:

010 = a műveleti kód után egy szó áll
011 = a műveleti kód után két szó áll
100 = a műveleti kód után nincs paraméter

Az utasítás leírása:

E művelet végrehajtásának megkezdésekor a processzor megállapítja, hogy az állapotregiszter megfelel-e a megadott feltételnek. Ha igen, akkor a processzor ennek az utasításnak a végrehajtása után megkezdi a 7-es számú megszakítás utasítássorozatának feldolgozását. Ez úgy megy végbe, hogy a processzor először tárolja az állapotregiszter és a programszámláló pillanatnyi tartalmát a supervisor-verembe. Ezután a $001C címen levő kettős-szót mint új programszámlálót betölti, és a feldolgozást ettől a címtől kezdődően folytatja.

Ha ennek a műveletnek a megkezdésekor a feltétel nem teljesül, akkor a feldolgozás a következő művelettel folytatódik.

Feltételként ($_{cc}$) a következő 16 bitkombináció valamelyike adható meg:

Név	**Kód**	**Jelentés**	**Név**	**Kód**	**Jelentés**
CC	0100	Carry = 0	LS	0011	nem nagyobb
CS	0101	Carry = 1	LT	1101	kisebb
F	0001	0	MI	1011	negatív
EQ	0111	egyenlő	NE	0110	nem egyenlő
GE	1100	nagyobb/egyenlő	PL	1010	pozitív
GT	1110	nagyobb mint 0	T	0000	1
HI	0010	nagyobb	VC	1000	túlcsordulás = 0
LE	1111	kisebb/egyenlő	VS	1001	túlcsordulás = 1

E feltételek pontosabb leírását, különösképpen a kapcsolóbitekkel való ábrázolásukat illetően e fejezet elejére utalunk.

TRAPV

V: PC – : -(SSP); SR – : -(SSP)
(vektor$_{\text{szám}}$ 7) – : PC

szoftveroldali megszakítás, ha a túlcsordulás magas
TRAP on overflow

Művelet	**Assembler szintaxis**
TRAPV	TRAPV
Operandus hossza:	nincs operandus

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	0

Az utasítás leírása:

E művelet végrehajtásának megkezdésekor a processzor megállapítja, hogy a túlcsordulás- (oVerflow) bit magasra van-e állítva. Ha igen, akkor a processzor ennek az utasításnak a végrehajtása után megkezdi a 7-es számú megszakítás utasítássorozatának feldolgozását. Ez úgy zajlik, hogy a processzor először tárolja az állapotregiszter és a programszámláló pillanatnyi tartalmát a supervisor-verembe. Ezután a $001C címen levő kettős-szót mint új programszámlálót betölti, és a feldolgozást ettől a címtől kezdődően folytatja.
Ha ennek a műveletnek a megkezdésekor a V kapcsolóbit nincs magasra állítva, akkor a feldolgozás a következő művelettel folytatódik.

egy byte vizsgálata
TeST byte

Művelet	**Assembler szintaxis**
TST "ea"	TST "ea"
Operandus hossza:	byte, szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
–	*	*	0	0

X nem változik.
N értéke 1 lesz, ha a vizsgálandó byte negatív volt, egyébként törlődik.
Z értéke 1 lesz, ha a vizsgálandó byte nulla volt, egyébként törlődik.
V törlődik.
C törlődik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	1	0	1	0	hossz		effektív cím					

hossz	ez a mező az operandus hosszát határozza meg: 00 = byte 01 = szó 10 = kettős-szó
effektív cím	a vizsgálandó byte címét tartalmazza. Az **adatok * változtatható** kategóriák valamennyi címzésmódja megengedett.

Az utasítás leírása:

Az utasítás a megadandó byte-ot az előjelre, ill. a nullával való egyezőségre vonatkozóan megvizsgálja, és a kapcsolóbiteket ennek megfelelően állítja.

helyi adattér felszabadítása
UNLinK local area

Művelet	**Assembler szintaxis**
UNLK Ax	UNLK Ax
Operandus hossza:	byte, szó vagy kettős-szó

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	1	1	1	0	0	1	0	1	1	Ax reg.		

Ax reg.	az Ax annak a címregiszternek a számát tartalmazza, amelyik helyi bázisként szolgál.

Az utasítás leírása:

Ez a LINK Ax utasítás ellentetje. Arra alkalmas, hogy a veremben a LINK által meghatározott lokális adatteret ismét felszabadítsa, és a veremmutatónak újra az legyen az értéke, ami a LINK utasítás előtt volt.
Ehhez először a megadott címregiszter átmásolódik a veremmutatóba, hogy ezután a veremben levő legfelső kettős-szót a megadott címregiszterbe lehessen tölteni.

tömörített BCD számok visszaállítása normál BCD számokká
UNPacK BCD
CSAK A 68020-AS PROCESSZORNÁL

Művelet	**Assembler szintaxis**
UNPK Dx-et Dy után offszettel	UNPK Dx, Dy, #"adat"
UNPK -(Ax)-et -(Ay) után offsz.-el	UNPK -(Ax), -(Ay), #"adat"
Operandus hossza:	nincs operandushosszúság megadva

Kapcsolók:

X	N	Z	V	C
–	–	–	–	–

X nem változik.
N nem változik.
Z nem változik.
V nem változik.
C nem változik.

Az utasítás formátuma:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	0	0	Ry reg.			1	1	0	0	0	R	Rx reg.		

Ry reg.	akár adat-, akár címregiszter lehet. Célként kerül felhasználásra.
R	azt határozza meg, hogy az adatokat adatregiszterekből kell-e elővenni, vagy a tárból, indirekt mutatókon keresztül: 0 = adatregiszter 1 = címregiszter
Rx reg.	ez a forrásregiszter regiszterszámát tartalmazza.

Az utasítás leírása:

Ez az utasítás a PACK utasítást „hatástalanítja". Ez egy forrásbyte két félbyteját (nibble-jét) egy szóra osztja el.
Forrás- és célregiszterként szolgálhat vagy két adatregiszter, vagy két címregiszterszám, amikoris a cél és a forrás címe ezeknek a regisztereknek az előzetes dekrementálásából adódik.
Először a két félbyte (nibble) szétválasztására kerül sor, a magasabb helyiértékű félbyte, tehát a 7...3 bitek a 11...8 bitekbe másolódnak át, és ennek az így létrejött szónak a 7...3 bitjei törlődnek.
Ezt követően a megadott offszet hozzáadódik az így előállított szóhoz. Ez az eredmény a célregiszterben kerül elhelyezésre.
Ha a forrás és a cél a tárban van, akkor az utasítás végrehajtása az alábbiak szerint történik:
A processzor a forrásként megadott szót beolvassa és kibővíti oly módon, hogy a 7...3 bitek a 11...8 bitekbe kerülnek, és a 7...3 bitek törlődnek. Ezután ehhez a szóhoz hozzáadódik az offszet, és az eredmény a tárnak arra a címére kerül, ahová a célregiszter mutatója mutat.

7. A 68000-ES PROCESSZOR AZ OPERÁCIÓS RENDSZEREKBEN

Egy számítógép szoftveroldali „lelkét" az operációs rendszere alkotja. Ez az operációs rendszer az egyszerű monitorral felszerelt terminálokból kezdve, az ugyancsak egyszerű, egymunkahelyes felhasználói rendszereken keresztül, egészen a többmunkahelyes, többfeladatú (Multi-User-Multi-Task) operációs rendszerekig terjedhet.

Ezek mindegyikében közös az a szoftver, amely a számítógépet a bekapcsoláskor elindítja: az a folyamat, amelyet általában „BOOT"-nak neveznek, és ami azt akarja kifejezni, hogy egy rendszert az egyes teljesítményfokozatok növeléséhez „alulról felfelé" töltenek be. Ez a betöltés a nagyszámítógépes rendszereknél órákat vesz igénybe, a mikroszámítógépek mai generációinál az operációs rendszerek beolvasása és indítása mindössze néhány percet, miközben még egy rövid rendszervizsgálatra is sor kerül.

Az operációs rendszer feladata tulajdonképpen az, hogy a számítógépet a bekapcsolást követően olyan, ellenőrizhető állapotba helyezze, amely lehetővé teszi, hogy a használó utasításokat adhasson a rendszernek. Az operációs rendszer megvizsgálja, hogy ezek az utasítások helyesek, és végrehajthatók-e, ill. hogy adott esetben nem ütköznek-e valamilyen korlátba. Ha a vizsgálat nem talált hibát, akkor az utasítást vagy végrehajtja, vagy végrehajtását egy külön folyamattal inicializálja, és felügyeli.

Utasításértelmező tulajdonságai mellett az operációs rendszernek fontos funkciója, hogy a programok számára olyan, fix interfészeket bocsát rendelkezésre, amelyeken keresztül ezek a programok egymás között adatokat tudnak cserélni.

Természetesen mindegyik program tartalmazhatna több vagy kevesebb, apróbb rutint, amelyekkel a program a felhasználótól (pl. billentyűzeten keresztül) adatokat fogadhatna, és ezeket vagy a saját adatait képernyőre vagy nyomtatóra küldhetné. Ezeknek a rutinoknak viszont ilyenkor nemcsak az említett perifériák interfészeihez kell illeszkedni tudniuk, hanem – és főként – arra is alkalmasnak kell lenniük, hogy az olyan tömegtárakból, ill. tömegtárakba, mint a hajlékony vagy merev lemez, olvasni, ill. írni tudjanak.

Bizonyára egyszerűen belátható, hogy a programozó munkáját lényegesen megkönnyíti, ha ezek a programrészek meghívható rutinokként már eleve a számítógép tárában vannak. Így a programozónak akkor, amikor a lemezen tárolt adatokra van szüksége, nincs más teendője, mint hogy a megfelelő rutint a megfelelő értékekkel meghívja. A dolog többi részét aztán már ezek a kész programrészek intézik. Amikor ezek a rutinok visszatérnek a meghívójukhoz, egy állapotjelzővel jelzik, hogy a művelet végrehajtása sikeres volt vagy sem, és sikeres végrehajtás esetén valamilyen formában szolgáltatják is a kért értékeket.

Az egyedüli probléma itt a helyes értékek megadása. A számítógépes rendszerek gyártóit a fejlesztés során piacpolitikai okok ugyanis olyan megfontolásokra késztetik, hogy a számítógépük lehetőség szerint csak a saját gyártmányú perifériákkal, pl. nyomtatóval működjön. Másrészt a hardvergyártók általában abban érdekeltek, hogy a készülékükön bármely, meglevő program futtatható legyen, hogy ily módon a szoftverfejlesztés tekintélyes költségeit megtakaríthassák, és egyszersmind ezeket a programokat a hardvervásárlás mellett szóló érvként is felhasználhassák.

Ahhoz továbbá, hogy ezeket az általános jellegű programokat a különböző hardvereken futtatni lehessen, a hardvertől esetleg olyan konfigurációt is el kell várni, ami gyakran csak nehezen valósítható meg, nem beszélve arról, hogy ez az új, jobb műszaki megoldások alkalmazásának akadályozójává is válhat.

Nézzünk például egy olyan programot, amely azt feltételezi, hogy ha a lemezről egy blokkot kér be, akkor 128 byte-ot fog kapni. Ilyen esetben a hardvergyártók számára csupán egy újabb hardver alkalmazása nem teszi lehetővé, hogy az egyes blokkok kapacitásának pl. 256 byte/blokk kapacitásra való növelésével a tárkapacitás is megnövekedjen. Ilyenkor jut szerephez az operációs rendszer, amelynek a meglevő hardverből az általánost absztrahálnia kell.

Ha a programnak a bevitel jelzésére egy bizonyos jelet kell szolgáltatnia, akkor a program ezt egy megfelelő művelettel az operációs rendszer tudomására hozza. Az operációs rendszernek most az a feladata, hogy ezt a jelet beolvassa, ill. egy, már a beviteli pufferben levő jelet a meghívó egységhez továbbítson. Ennek során tehát a program számára teljesen mindegy, hogy a jelet melyik eszközről kapja; ugyanúgy lehet ez akár egy párhuzamos, akár egy soros adatbusz is.

Az operációs rendszernek – a perifériákhoz hasonló módon – a tömegtáraknál is el kell végezni az absztrakciót. Így a rendszer a programozói interfészében olyan utasításokat bocsát a felhasználó rendelkezésére, amelyek segítségével meghatározott számú blokkokat lehet olvasni és írni. Ennél az interfésznél egyedüli kötöttség mindössze az operációs rendszer által szoftveroldalról feldolgozandó jelek száma. Ha a felhasználói program olyan, hogy az blokkonként 128 jelet olvas, a lemezen viszont hardveroldalról blokkonként 512 jel kerül feldolgozásra, akkor az operációs rendszernek a feladata, hogy a megfelelő blokkot, és azon belül a megfelelő 128 jelet megtalálja, elolvassa vagy felülírja.

Az elmúlt években a mikroszámítógép-iparban bekövetkezett rendkívül gyors fejlődés következményeként a 70-es évek végén elárasztották a piacot az olyan számítógépek, amelyek mindegyike valamiféle saját felhasználói interfésszel rendelkezett. Ez aztán a programok tökéletes káoszához vezetett.

Az egyes rendszerek közötti inkompatibilitásokból néhány cég hasznot tudott húzni, míg mások, akik nem bírták elegendő „szusszal", az ismertebb rendszerekkel szembeni inkompatibilitásuk miatt gazdaságilag csődbe jutottak. Csak a 70-es évek végén kezdett el a kisszámítógépek piacán egy operációs rendszer egyre inkább tért hódítani. Ez az operációs rendszer a DIGITAL RESEARCH által kifejlesztett

CP/M

jelű rendszer.

A CP/M mikroszámítógépes operációs rendszer (Control Program for Microcomputer) egy konzolon, egy író interfészen, egy olvasó interfészen és egy nyomtató csatlakozón kívül csak 16 hajlékonylemezes meghajtókészülék csatlakoztatásának a lehetőségét biztosította.

A CP/M és a felhasználói program közötti interfészként a CPU címterületének a legalsó 128 byte-ja szolgált, és két helyen volt ugrási pont az utasítások átadásához. Az operációs rendszer úgynevezett BIOS (Basic Input-Output System) része képezte a tulajdonképpeni hidat az operációs rendszer és a hozzá csatlakoztatott, speciális perifériák között.

Az operációs rendszerek fejlesztésében természetesen nem a CP/M volt az első lépcső. Az első tapasztalatokat a matematikusok és fizikusok már az első, programozható számítógépek kapcsán megszerezték.

Az első mikrogépek kifejlesztésének idején már egy sereg érdekes operációs rendszer létezett. Közöttük voltak az olyan, ma már továbbfejlesztett formában megtalálható operációs rendszereknek az elődei, mint pl. a Bell Laboratories, AT & T UNIX nevű rendszere.

A nagyszámítógépeknél a drága hardver miatt, gazdasági okokból lehetővé kellett tenni, hogy a számítógépet egyidejűleg többen is használhassák. Azonban ennek a megvalósítása is meglehetősen költséges, ha azt akarjuk, hogy ezen keresztül a felhasználó munkáját támogassuk, és ne akadályozzuk. Nagy probléma például, ha egy programnak esetleg egy időben két felhasználó számára kell „futnia". A megoldás egyszerű lenne, ha a programot a tárban el lehetne tolni, azaz a program számára mindegy lenne, hogy a tár melyik részében fut.

A programozónak azonban többnyire nincs lehetősége arra, hogy a kódokat és az adatokat a tárban teljesen szabadon tologathassa. A program általában a tár egy meghatározott címtartományához van kötve, és külön segédeszköz nélkül nem bontható meg.

7.1 A tárkezelő egység (MMU)

Ha viszont ezt a programot egyidejűleg több felhasználó akarja használni, akkor a számítógép tárában ugyanannak a címtartománynak egyidejűleg kétszer kell meglennie.

Ahhoz, hogy a processzor és a többi berendezés – amelyek a tár azonos címeit használják – egymás működését ne zavarják, egy speciális egységet kell a rendszerbe iktatni.

Ez a tárkezelő egység – amelyet az eredeti, angol nevéből, a Memory Management Unit-ból származtatva röviden MMU-nak nevezünk – egyszeres vagy kétszeres címeltolást tud végrehajtani.

Először a teljes írható-olvasható tárat megfelelően sok kicsi, egymással azonos nagyságú blokkokra, az úgynevezett lapokra (pages) osztja fel. A következőkben bemutatásra kerülő példában ehhez a CPU 24 címvezetéket bocsát rendelkezésre, amelyek 0-tól 23-ig vannak számozva.

Az alsó 8 címvezeték közvetlenül a tárra van kapcsolva, így ezek egy tetszőleges lapon belül 256 címet tudnak megkülönböztetni.

A felső 16 címvezeték az MMU-ra van kapcsolva, amely így a hozzá érkező jelekkel 65 536 különböző lapot tud kiválasztani.

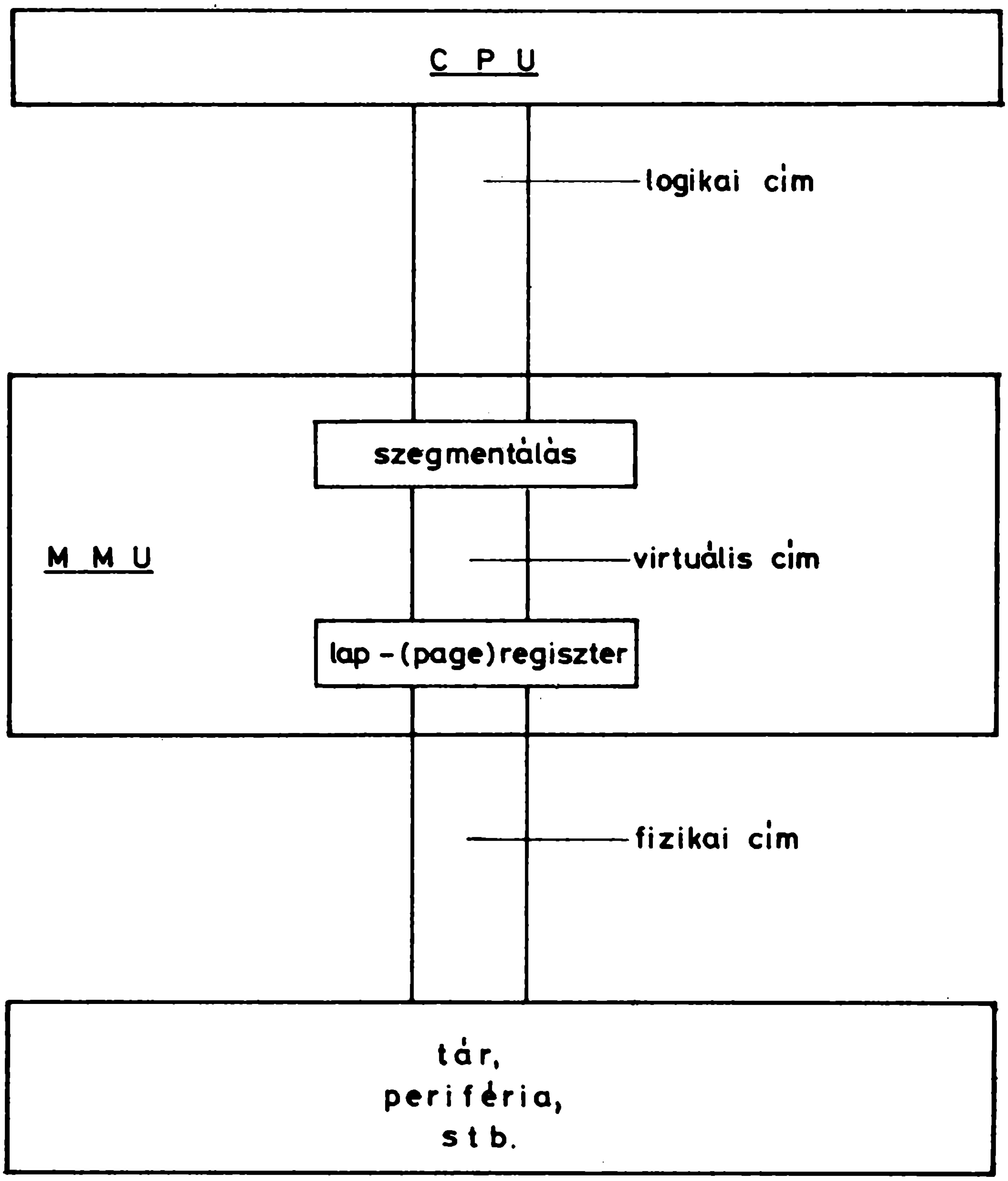
Ha egy műveletnek a tár egyik ilyen lapjára van szüksége, akkor ezt az operációs rendszertől „kéri el”. Az operációs rendszer normál esetben verem szervezésű, a tár szabad részére vonatkozó jegyzéket vezet, amelyből kiveszi a legfelső szabad blokkot. Ennek a blokknak (lapnak) a címe most betöltődik az MMU egyik konverziós regiszterében úgy, hogy ebben az összehasonlító táblában a felhasználói program által kért cím az itt levő oldal kezdőcímére mutat.

Ezzel a módszerrel a tárat a laphatárokon belül tetszőlegesen át lehet szervezni, és a processzor ugyanazon virtuális címeit különböző, fizikailag is létező címekre át lehet másolni.

A lapokra való osztás mellett lehetőség van szegmentálásra is. Ez azt jelenti, hogy a rendszerhatárokon belül egy szegmensbe tetszőleges számú lap fogható össze. Ily módon a címek háromszoros felosztásáról (és kétszeres átalakításáról) beszélhetünk:

A logikai cím az a cím, amit a processzor kiad. Ha pl. egy folyamat egy adatot akar betölteni a 012 345 abszolút címről, akkor éppen ez az a cím, amit a processzor a címbuszra ad.

Erről a logikai címről – nem csak ebben a példában – a legalsó 8 címbit nem kerül figyelembevételre, mert ezek csak a véglegesen megtalálandó oldalon levő offszetet határozzák meg.


C P U
logikai cím
szegmentálás
M M U
virtuális cím
lap - (page) regiszter
fizikai cím
tár,
periféria,
stb.

Az MMU működésének elve egy számítógéprendszerben

Az MMU most azt vizsgálja meg, hogy a logikai cím felső 16 bitje a folyamat rendelkezésére álló szegmensek határain belül van-e. Ha a 0123xx cím a határokon kívül esik, akkor megfelelő hibajelzés történik. A 68000-esnél buszhibakizárás kerül kijelzésre.

Ha a 0123xx cím a választott határokon belül van, akkor az MMU ezt követően a hozzáférési jogokat vizsgálja. Ezeket a jogokat egy szegmens kialakításakor az MMU közli az operációs rendszerrel, és általában a következők szerint csoportosíthatók:

Read-Only (csak olvasás)	A szegmensből csak olvasni szabad. Az egyes MMU-któl függően ezen adatok utasításokként való végrehajtását meg lehet tiltani.
SYSTEM-Only (csak rendszer)	Ha ennek a bitnek az értéke 1, akkor a használó ezekhez az adatokhoz nem férhet hozzá. Ez főként az operációs rendszer és annak változói szempontjából lényeges.
EXECUTE-Only (csak végrehajtás)	Ezzel az dönthető el, hogy az ebben a tartományban levő adatokat a műveleti kód elővételekor csak a CPU olvashatja-e. A program biztonságának ellenőrzése mellett ez a védőmechanizmus azért került beépítésre, hogy arra az esetre, ha a használó számára meg kell engedni, hogy a programot ettől a helytől kezdődően futtassa, a jogtalan olvasást (és másolást) meg lehessen akadályozni.
No Execution (nincs végrehajtás)	Ez a bit azt akadályozza meg, hogy a processzort ne tévessze meg az, ha a használói adatok valamilyen okból programként kerülnének interpretálásra.

Ha a hozzáférési jogokat illetően valamennyi feltétel teljesül, akkor az MMU a beérkezett 0123xx címet egy belső, virtuális címre képezi le. Ez a virtuális cím az előzőekben leírtak szerint kezelhető. Ellenőrzésre kerül, hogy ezen a virtuális címen a felhasználó számára rendelkezésre áll-e a központi tár egy lapja. Ha igen, akkor az MMU a megfelelő fizikai címet a tárhoz vezető „saját” címbuszára helyezi, és ezzel kezdődik meg a tárhoz való tulajdonképpeni hozzáférés.

Ha a szegmensen belül a lap (page) még, vagy már nem létezik, akkor ez ismét egy buszhibát vált ki, amit az operációs rendszernek kell feldolgoznia. Így pl. az operációs rendszer az igényelt lapot „elkérheti” a tár szabad részét kezelő egységtől, ennek címét közli az MMU-val, hogy ezután a tulajdonképpeni tervezett címhozzáférés megindulhasson.

Ez az eljárás nagyon egyszerű akkor, ha arról van szó, hogy a processzor által előállított adatokat kell a tárban elhelyezni. Ha viszont egy programrészhez kell hozzáférni, ami még nincs a tárban, akkor ez már lényegesen bonyolultabb. Ilyen esetben az operációs rendszernek először be kell olvasni a háttértárból a program megfelelő blokkját, majd csak ezt követően kezdeményezheti a hozzáférést, amire viszont a 68000-esnél sajnos még nincs lehetőség.

Rendkívül nehézzé válik egy sikertelen hozzáférés kijavítása abban az esetben, ha pillanatnyilag nem áll rendelkezésre szabad tárterület. Ilyenkor az operációs rendszer megkísérli, hogy egy, abban a pillanatban (remélhetőleg) használaton kívüli lapot a lemezre helyezzen el. Ezzel a SWAP eljárással az utóbbi években már behatóan foglalkoztak a tudósok, és ezzel kapcsolatban különböző megfontolásokat lehet tenni:

Gondot okoz pl. a következő eldöntése: vajon azt a lapot kellene-e felszabadítani, amelyet a rendszer a leghosszabb ideje nem használt, vagy talán pontosan azt, amit éppen a megelőző pillanatban használt?

Első ránézésre az első verzió tűnik a legésszerűbbnek; a lap már régóta nincs használatban, tehát a feldolgozása már befejeződhetett. De az is lehet, hogy talán éppen ezért van régóta „parlagon”, mert a feldolgozásához szükséges folyamatot eddig más folyamatok elnyomták, és az éppen feldolgozás alatt levő feladat elvégzése után tud a CPU ennek a lapnak a munkába vételére időt szakítani? Ebben az esetben az éppen felszabadított lapot rögtön újra be kellene tölteni.

A második verzió, tehát a tár éppen utoljára feldolgozott részének a felszabadítása szintén nem problémamentes.

Fusson pl. egy rendszerben három folyamat, ebből kettő legyen azonos prioritású, a harmadik alacsonyabb. A CPU idejének igénybevételéért folyó küzdelemben tehát ez a harmadik a „leggyengébb”.

Ha most a két, azonos prioritású folyamat közül az egyik egy új laphoz kíván nyúlni, de pillanatnyilag nincs szabad lap, akkor az operációs rendszer statisztikailag leggyakrabban az azonos prioritásút tenné félre, mert minden bizonynyal ez az a folyamat, amelyet az éppen futó folyamat megszakított. A legésszerűbb persze az lenne, ha az alacsonyabb prioritásút tenné félre, hiszen ennek amúgy is várakoznia kell. Itt tehát az idő kezelése mellett még a prioritásokat is kezelni kellene. Ez a rendszert tovább bonyolítaná, és főleg az úgynevezett „sceduler”-nek, az operációs rendszer folyamatváltásokért felelős részének a működését lassítaná le.

Egy fontos dolgot eddig még egyáltalán nem vettünk figyelembe:

Ha egy éppen felszabadítandó lap tartalma a beolvasása óta nem változott, akkor azt nem kell újra a lemezre írni, mivel az már úgyis rajta van, vagy egyébként is üres volt. Ha viszont időközben módosult, akkor ezt a körülményt – igény esetén – minden jobb MMU egy MÓDOSULT bit segítségével jelzi a CPU-nak.

Ha most a kért virtuális című oldal megvan, és így végre a fizikai tárcím a buszra került, akkor megkezdődik a tárhoz való tulajdonképpeni hozzáférés. Ilyenkor is felléphetnek különböző hibák, amelyek attól kezdődően, hogy egy egység egy bizonyos időn belül nem válaszol, az adatok olvasásakor bekövetkezhető paritáshibákig terjedhetnek.

Mindezek a hibák a 68000-es rendszereknél buszhiba jelzéshez vezetnek. A 68000-es még nem képes arra, hogy a buszhibák nagyobb része után a működését ismét helyes módon folytassa, úgyhogy ilyenkor megfelelő programmódosításokat kell eszközölni. A 68010-es processzor azonban, amelynél a kizárások feldolgozása már javított formában történik, lehetővé teszi az operációs rendszer számára ezt a funkciót.

7.2 Az operációs rendszerek támogatása

A 68000-es processzor az olyan, rugalmas, többmunkahelyes operációs rendszerek, mint a már említett UNIX és ehhez hasonlók támogatását illetően a 68000-es család messze leggyengébb tagja.

A későbbi testvérprocesszorokkal azonos módon támogatja ugyan a USER/SUPERVISOR koncepciót, ami az operációs rendszer programozója számára megkönnyíti, hogy a rendszerét kezelői vagy programozási hibákkal szemben megvédje. Az elegendő számú, általánosan használható regiszterek (úgynevezett general purpose registers) a programozó számára arra is lehetőséget adnak, hogy hatékonyan, áttekinthetően, és mindenekelőtt modulárisan programozhasson, de arra nem, hogy pl. egy buszhiba után a processzor a működését ismét helyesen folytatni tudja.

Akkor is nehézségek merülnek fel, ha a processzornak egy szimulációs folyamatot kell feldolgoznia, mert a processzor állapotregiszterének a supervisor részét mindig csak a normál módon lehet kiolvasni. Így aztán egy szimulátoron belüli nyomkövetésnél (trace) nehéz dolog egy, az eredetitől eltérő tartalmú supervisort szimulálni.

A sebességet illetően a 68010-es processzor a prefetch technikájának köszönhetőne a 68000-est felülmúlja. Ez a technika különösen a DB_{CC} általi loop műveleteknél használható előnyösen (lásd 5. Az utasításkészlet c. fejezet bevezetőjét és a 8. Programozási példák c. fejezetet).

Emellett a 68010-esnél bevezették a vektor-bázis-regiszter technikáját. E regiszter segítségével meghatározható, hogy mely címen kezdődjön a 68010/68020-as vektortáblája. Ez akkor előnyös, ha különböző feldolgozási folyamatoknak különböző vektorokra van szükségük. Mindegyik folyamat kaphat a saját feldolgozási folyamatához, lemezhozzáféréshez és hasonlókhoz általában szükséges mutatókból (pointer) egy-egy készletet, amelyekkel a szabad vektorokat tetszés szerint átprogramozhatják anélkül, hogy egymás között körülményes „alkudozást" kellene folytatniuk. Gyakran kell pl. egy órajeladó egység ütemidejét ciklikus megszakítás kiváltására használni. Ezt most így mindegyik feldolgozási folyamat igénybe vehető saját céljaira, pl. arra, hogy felismerje azt, ha egy rendszer „zsákutcába" jutott, és az akadályt el tudja hárítani.

Nyilvánvalóan ezek az alkalmazások ilyenkor a legritkább esetben valós idejű feldolgozások, mivel hogy pl. az említett ütemjeladót valamennyi feldolgozási folyamat használni tudja.

Az operációs rendszer feladata itt az, hogy egy újonnan felépítendő feladathoz a standard vektorokat inicializálja, a táblán a különböző kezdőcímeket archiválja, és ha egy új feldolgozási folyamatra kerül sor, akkor az új feladat „saját vektorát írja be a vektor-bázis-regiszterbe (VBR).

Amint azt a 68020-as regiszterszerkezetének ismertetésénél már láttuk, a 68020-asnak összesen három veremmutatója van, ezek közül kettő a SUPERVISOR állapotra, egy pedig a USER állapotra vonatkozik.

A USER veremmutató szemantikája a megelőző modellekéhez képest nem változott; ugyancsak változatlan maradt a SUPERVISOR – most MASZTER – veremmutatóé is. Újként került be a megszakítási veremmutató (ISP), ami akkor jut szerephez, ha egy tetszőleges kizárási feltétel áll fenn.

A veremmutatók hármas felosztása ésszerű, mert most a folyamatok valamennyi rezidens adata a MASZTER veremben lehet, míg a megszakítási veremben az a normál üzem folyik, ami egyébként máskor az egyetlen SUPERVISOR veremben szokott folyni. Ha változtatni kell a feladatot, akkor ezt a megfelelő megszakítás bejegyzi a megszakítási verembe. Most az operációs rendszer részéről a veremmutató megváltoztatható, hogy a futó feladat pillanatnyi állapotát a MASZTER verem egy meghatározott helyére lehessen tárolni. Ezután a MASZTER veremmutató áthelyeződik az elindítandó folyamat CPU-ban levő címére. Egy MOVEM művelet (MOVE Multiple registers) után megtörténik a regiszterek cseréje. Ha most a megszakítási veremmutatót vissza kellene cserélni, akkor csak a megszakítási veremben levő legutolsó bejegyzést kellene az új eredmény címével kicserélni ahhoz, hogy a feldolgozási folyamatok cseréje teljes legyen.

Annak ellenére, hogy meglehetősen nehezen érthető, és a programozás során valószínűleg hibákat is okozhat, figyelemre méltó az a tény, hogy az egyes feladatok váltásának teljes ideje alatt a megszakítások aktiválhatók maradhatnak. Ha egy megszakítás lép fel, akkor ez csak a megszakítási verem hosszát növeli, a maszter vermet viszont nem befolyásolja. Ha az egyes feldolgozási folyamatok váltogatása közben egy megszakítás lépne fel, akkor ennek minden bizonnyal valami súlyos oka lehet, de a 68020-asnak ilyen hibák gyors elhárítására is vannak lehetőségei.

A fentiekben vázolt elgondolás nemcsak ennél a processzornál létezik, hanem már a 68000-esnél is megvolt.

Itt most a TAS (Test byte And Set always bit 7) bytevizsgáló és átkapcsoló utasítást ismertetjük, amelyet általában szemafor műveletnek szokás nevezni.

Az operációs rendszertechnikában azokat a jeleket nevezik szemforoknak, amelyek azokkal, akik rájuk figyelnek, azt tudják közölni, hogy egy kívánt hozzáférés megengedett-e vagy nem.

Ezt a problematikát egy rövid példán mutatjuk be:

Tegyük fel, hogy egy rendszerben legkevesebb két olyan feldolgozási folyamat van, amelyek egy óra által mutatott időhöz akarnak hozzáférni. Az egyik folyamat dolga az óra beállítása, az összes többi csak olvashatja azt. Tegyük fel, hogy az általunk figyelt időpontban az óra 14 : 59-et mutat. Most az első folyamat a percektől indulva, elkezdi az óra leolvasását. Az első információ az 59. Most az órát állító folyamat ezt a megelőző folyamatot megszakítja, az időt 15 : 00-ra módosítja, és ismét az olvasó folyamatot engedi „szóhoz" jutni. Ez most már persze 15 : 00-át „lát". Amit leolvas, az tehát 15 : 59 lesz. Az ilyen, az időtől függő vezérlések következményei az Olvasó fantáziájára vannak bízva.

Ilyen esetben segít az úgynevezett szemafor, amelynek állására mind az olvasást, mind az írást végző folyamatnak figyelnie kell:

Az olvasást végző folyamat például a következő utasítássorozat lehet:

```
                          •
                          •
várj a nulla olvasásra:   TAS       ÓRA
                          BNE       várj a nulla olvasásra
                          •
                          •
                          •         ; olvasd az időt
                          •
                          CLR.B     ÓRA
                          •
                          •
```

Ha feltesszük, hogy az ÓRA 7. bitje nulla, akkor a Test And Set utasítás a Z kapcsolóbitet és az ÓRA 7. bitjét magasra állítja (értékük 1 lesz).

Ha az olvasási folyamat itt egy magasra állított Z kapcsolóbitet észlel, akkor azt gondolhatja, hogy abban a pillanatban csak ő férhet hozzá az órához. Ha ezt a folyamatot az óra olvasása közben valamely helyen az írást végző folyamat megszakítaná, akkor ez utóbbinak egy azonos rutinon kell áthaladnia:

```
                          •
                          •
várj a nulla írására:     TAS       ÓRA
                          BNE       várj a nulla írására
```

```
●
●          ; állítsd be az időt
●
●
CLR.B      ÓRA
●
●
```

Itt azonban a TAS utasítás nem állítja a Z bitet, mivel az ÓRA 7. bitje még mindig magas. Az írást végző folyamat tehát mindaddig végrehajtja a TAS...BNE...TAS ciklust, míg az ideje lejár, az olvasást végző folyamat befejezi az olvasást, és az ÓRÁ-t ismét törli. Ha most behívásra kerül az írást végző folyamat, akkor a TAS már átkapcsol, és az órát a beállítás befejeződéséig automatikusan „megvédi" mindenféle más olvasási vagy írási folyamattal szemben.

A bemutatott rutinok alkalmasint arra példák, hogy ezeket hogyan nem kell csinálni, több okból is: először is az olvasás kizáró rutinja úgy van felépítve, hogy egy olvasást végző folyamat az összes többi, ugyancsak olvasni kívánó folyamatot kizárja. Ez viszont fölösleges, mert az órát egyidejűleg több, különböző folyamat is olvashatná anélkül, hogy ezek egymást zavarnák. Olvasáskor csak az írási folyamatot kell kizárni, ill. az olvasást arra az időre, amíg az írást végző folyamat az úgynevezett kritikus szakaszban van.

Másrészt az a módszer, hogy a szemafor szabad jelzésére várakozó folyamatot a foglaltsági-várakozási (busy-waiting) ciklusban futtassuk, minden jobbfajta operációs rendszer ismeretének ellene mond. Azért, mert egy folyamat valamiből történetesen ki van zárva, illene még legalább egy másik, de leginkább azt a folyamatot futni hagynia, amely az „ő" szemforát blokkolta.

Ezzel a kis kitérővel, amit az operációs rendszerek mellékfolyamatai szabályozásának területén tettünk, ezeknek az utasításoknak csak egy elvét kívántuk szemléltetni: ezek oszthatatlan műveletek, más szavakkal a bit vizsgálata és az ezt követő kapcsolása között egyetlen más egység sem férhet a buszhoz. Erre a megszorításra feltétlenül szükség van, mert az nem elegendő, hogy a művelet végrehajtása közben a megszakítások nem kerülnek figyelembevételre (ezt egyébként egy modern CPU már önmagában is biztosítja), hanem mindenekelőtt a buszon levő, esetleges további processzorokat és buszmasztereket kell az éppen aktív szemfornak távol tartani.

Az operációs rendszertechnika e területe iránt érdeklődő olvasóknak az Irodalomjegyzékben felsorolt műveket ajánljuk figyelmébe, amelyek az ilyen jellegű problémákhoz megoldásokat, vagy legalábbis a megoldáshoz ötleteket javasolnak.

7.3 A programozási nyelvek támogatása

Az operációs rendszerek mellett a programozási nyelvek is függenek a processzor teljesítményétől. Elvileg ugyan bármely processzorhoz bármely interpreter vagy compiler használata elképzelhető, azonban a lehetőségek mellett itt a gyakorlati hasznosság is fontos szerepet játszik. Így pl. értelmetlen lenne egy SIMULA-compilert írni egy soros, 8 bites processzorra, mivel ez már egy közepes nagyságú programnál elérné a címezhetőségének határait.

A 68000-es sorozat processzoraihoz azonban mind az alacsony, mind a közepes és magasszintű programozási nyelvek használhatók. A kedvezően megválasztott számú regisztereknek köszönhetően gyorsan és áttekintően írhatók assembler programok, de a 68000-esek egyes utasításai magasabb szintű nyelvek, mint a C, a Pascal, a COBOL, vagy a régóta ismert FORTRAN és ALGOL, valamint az igazán új, részben interpreter nyelvek, mint a LISP és PROLOG irányába mutatnak.

Mindezen nyelvek fordításánál problémát jelent a paraméterek átadásának értelmes formában való megvalósítása. A 68000-es erre számos lehetőséget kínál, melyek közül egyszerűsége és gyakorlati alkalmazhatósága következtében az egyik kiválik a többi közül: ez a vermen keresztüli paraméterátadás.

Ehhez egy folyamat meghívása előtt valamennyi, a folyamat számára szükséges értéket a normál verembe kell helyezni.

Például a test (x, y, z) kifejezés egy olyan assembler kódot állít elő, ami így nézhet ki:

```
●
●
MOVE.L   z ,–(SP)
MOVE.L   y ,–(SP)
MOVE.L   x ,–(SP)
CALL     test
●
●
```

Vegyük észre, hogy elsőként az utolsó paraméter kerül a verembe. Ez azért van így, hogy aztán a paraméterek ismét a helyes sorrendben kerüljenek a tárba, mert mint ismeretes, a vermek felülről lefelé „töltődnek".

A „test" rutinba való belépéskor a verem tehát így néz ki:

magasabb cím	z
	y
	x
SP	visszaugrási cím
alacsonyabb cím	

Ha most a „test” rutinban egy értéket kellene elővenni a veremből, akkor elvileg csak

```
az   x  értéket a  MOVE.L   4(SP),D0
az   y  értéket a  MOVE.L   8(SP),D1
és a z  értéket a  MOVE.L  12(SP),D2
```

utasítással kellene betölteni. Ha azonban még további adatokat kellene a verembe helyezni, ha tehát pl. egy regisztert kellene tárolni, akkor a veremmutatóval való számítás igen körülményes lenne. Ennek elkerülésére került bevezetésre az úgynevezett helyi bázis, ami a paraméterek számára egy fix vonatkoztatási pontot képvisel.

Erre a célra hozták létre a LINK műveletet. Ennek a meghívása egy címregiszter nevével és egy abszolút értékkel végezhető.

A címregiszter neve (tehát a száma) azt a regisztert jelöli ki, amelyik ettől a pillanattól kezdve a helyi bázist jelenti. Először ennek a tartalma tárolása kerül a verembe, ezután a veremmutató pillanatnyi állása a helyi bázisban töltődik, majd végül a veremmutató értéke az abszolút, második értékkel csökken. A veremmutató értékének ez a csökkentése egy „szabad” teret hoz létre a helyi bázis és a veremben levő esetleges bejegyzések között. Ez a tér szolgál a helyi változók elhelyezésére.

Legyen a „test” rutinban két, kettős-szó hosszúságú változó deklarálva:

```
proc test (valt1, valt2, valt3) long valt1, valt2, valt3;
  long kozbossz1, kozbossz2;
  begin
  kozbossz1: = valt1 + valt2;
  kozbossz2: = kozbossz1 + valt3;
  kozbossz2;
  end
```

Erről egy ilyen fordítás készíthető:

```
test:     LINK          A6, # -8
          MOVEM.L       # tárolni, -(SP)
          MOVE.L        8(A6),DO
          MOVE.L        12(A6),D1
          ADD.L         D0,D1
          MOVE.L        D1,-4(A6)
          MOVE.L        16(A6),DO
          ADD.L         D1,D0
          MOVE.L        D0,-8(A6)
          MOVEM.L       (SP)+, #elővenni
          UNLK          A6
          RET
```

Ebbe a rutinba való belépéskor a veremben nincs semmiféle különleges érték:

	z
	y
	x
SP	vissztérési cím

A LINK A6,# −8 utasítás elsőként az A6 regiszter értékét a verembe helyezi, azután a veremmutató pillanatnyi értékét az A6 regiszterbe másolja, amelyik most a helyi bázis. Ezután a veremmutató értéke 8-cal csökken, ami a helyi bázis alatt két, kettős-szó hosszúságú változó részére foglal helyet:

	z
	y
	x
	visszatérési cím
A6	A6 tartalma
	szabad hely a kozbossz1-hez
SP	szabad hely a kozbossz2-höz

Ezután valamennyi, esetlegesen szüksége regiszter a verembe kerül. Ez a MOVEM utasítás egyszeri kiadásával történik. Ehhez meg kell adni valamennyi tárolandó regiszter jegyzékét, és célként a vermet. Ezután történik meg a lokális változókban levő közbenső összegek (itt egyébként fölösleges) összeadása. A MOVEM utasítás végrehajtása előtt tehát így néz ki a verem:

	z
	y
	x
	visszatérési cím
A6	A6 tartalma
	kozbossz1
	kozbossz2
	1. tárolt regiszter
	2. tárolt regiszter
	3. tárolt regiszter
	●
	●
	●
Sp	utolsó tárolt regiszter
	esetleges korábbi verembejegyzések
	(modulok meghívása, megszakítások és hasonlók)

A megfordított MOVEM utasítással a tárolt regiszterek újra visszakerülnek az eredeti állapotukba, hogy ezt követően az UNLK A6 (unlink) utasítással a vermet a visszaugráshoz felszabadítsák. Először betöltődik a veremmutatóba a helyi bázis értéke, aminek következtében a helyi változók megsemmisülnek, hogy ezután az A6 regiszter visszakapja azt az eredeti értékét, amit a LINK utasítás korábban a verembe töltött. Ezután megtörténik a szokásos visszaugrás; az esetleg visszaadandó paraméter a CPU egy vagy több regiszterében, vagy egy globális változóban van.

Egyszerűen belátható, hogy fix regiszterek és tetszőlegesen hosszú vermek kombinálásával lehetőség van rekurzív folyamatok feldolgozására. A rövid programozási példákat ismertető fejezetben egy rekurzív függvény ezt az eljárást használja.

A LINK/UNLINK e mechanizmusával az egyes kontextusok cseréje egyszerűen elvégezhető. Ezzel összefüggésben lényeges újítást jelent a 68020-as processzor modultechnikája.

7.4 A 68020-as processzor modultechnikája

Ez a processzor rendelkezik a kontextuscsere igen drága, de ugyanakkor rugalmas lehetőségével, melynél a hozzáférési jogosultságok vonatkozásában egyidejűleg bizonyos állapotváltoztatás is elvégezhető.

Ehhez a CALLM és a RTM utasításokat használja, amelyek a CALL Module és a Return from Module mnemonikjai.

Egy modul meghívásakor átadásra kerül egy moduldeszkriptor címe, amely deszkriptor a modult és annak értékeit írja le pontosan.

OPT	**Típus**	**Hozzáférési szint**	**Nullák**
modul program-mutatója			
modul adattartomány-mutatója			
modul veremmutató (opcionális)			
további, a felhasználó által meghatározandó információk			

Az OPT (opció) mező értéke 000 és 100 lehet, és azt határozza meg, hogy a meghívott modul az értékét a veremből várja (000), vagy hogy a meghívott modul az adatait egy mutatón keresztül, a meghívó modul vermében találja (100).

A típus a deszkriptor jellegét specifikálja, az értéke 00 vagy 01 lehet. Ha a típus értéke 00, akkor a hozzáférési jogban nem következhet be változás. Ha viszont az értéke 01, akkor a cél hozzáférési szintjének az információja kerül felhasználásra.

A modul program-mutató a meghívott modul első szavára mutat. Ez az első szó annak az adat- vagy címregiszternek a számát tartalmazza, amelyet a modulveremtartományban biztonságba kell helyezni azért, hogy ezután a moduldeszkriptor adatmutatóját fel lehessen venni. Az első végrehajtandó szót közvetlenül a rákövetkező szó tartalmazza.

A veremben egy úgynevezett modul-veremtartomány kerül kialakításra, amelybe a megfelelő értékek a következő módon kerülnek be:

SP+24	opcionális argumentumok		
SP+20	tárolt veremmutató		
SP+16	tárolt modul-adatmutató		
SP+12	tárolt programszámláló		
SP+ 8	moduldeszkriptor-mutató		
	foglalt		
	00000000	argumentum szám	
	00000000	kapcsolók (flags)	
SP	Opt	típus	tárolt hozzáférési szint

A hozzáférési szinteket a CPU saját maga, belül, nem tudja feldolgozni, de arra van lehetősége, hogy egy hozzáférésen belül a hozzáférési jogosultságban beálló változás helyességét egy külső hardverrel felülvizsgáltassa. Ha a meghívandó modul nincs feljogosítva arra, hogy a hozzáférési jogait kiszélesítse, akkor a külső logikai egység ezt megfelelő állapotával jelzi, ami a 68020-as processzornál egy formátumhiba kizárási feldolgozást indít el.

Míg ezeket a modulokat a veremben CALLM utasítással lehet felépíteni, a feldolgozást követően lebontásuk az RTM utasítással történik. Az előbb bemutatott veremtartományba való bejegyzések alapján a 68020-as felismeri, hogy milyen típusú volt a modul meghívása, és hogy hogyan kell visszaugrania a korábbi modulhoz.

Ez a technika kiválóan alkalmas például arra, hogy különböző SUPERUSER-eket úgy lehessen egy MMU-val kezelni, hogy ezek ugyan az elvi supervisor állapotukat megtartsák, de ne nyúljanak át feltétlenül más superuserek tártartományaiba.

Ennek a modultechnikának az ügyes alkalmazása – a társprocesszor-technikához hasonlóan – döntően a csatlakoztatott hardver „intelligenciájától" függ. Itt még további hardverek fejlesztésére kell számítani.

Az operációs rendszerekben a felhasználók és a feladatok cseréje is rendkívüli módon felgyorsítható a modultechnika alkalmazásával. Ott, ahol korábban az operációs rendszer szoftverével, az MMU-n keresztül kellett tartományokat felszabadítani és lefoglalni, most a 68020-as önműködően vezérli az MMU-t a felhasználó kívánságának megfelelően.

8. PROGRAMOZÁSI PÉLDÁK

Ebben a fejezetben két példán mutatjuk be a 68000-es utasításaival és címzésmódjaival kapcsolatos néhány különleges tulajdonságát.

Az egyik sajátosság az ABCD és az SBCD mnemonikokkal jelölt, BCD aritmetikai utasításokkal kapcsolatos. Ezek elvileg kétféle címzésmód alkalmazását engedik meg: vagy mind a cél-, mind a forrásoperandus egy-egy adatregiszterben van, vagy pedig mindkettőt a – (An) jelölésű, „címregiszter közvetett címzése előzetesen dekrementálással" nevű címzésmóddal lehet elérni.

Az az eset, amikor mindkét adat adatregiszterekben áll rendelkezésre, rendkívül egyszerű, és nem szorul további magyarázatra.

Mivel itt azonban mindig csak byte-ok dolgozhatók fel, ezeknél az utasításoknál a második, lassabb, de rugalmasabb címzésmódot is meg kellett engedni: a számok egy kezdőcímtől felfelé csökkenő címeken helyezkednek el a tárban úgy, mintha a számokat balról jobbra olvasnánk, miközben a címek balról jobbra nőnek.

Adjuk össze például az 55347726 és a 9876543321 BCD számokat.

A két érték a következőképpen helyezkedik el a tárban (byte-onként tekintve):

alacsonyabb cím	9	8
	7	6
	5	4
	3	3
magasabb cím	2	1
CÉLMUTATÓ →		

alacsonyabb cím	0	0
	5	5
	3	4
	7	7
magasabb cím	2	6
FORRÁSMUTATÓ →		

A CÉLMUTATÓ és a FORRÁSMUTATÓ olyan mutatók (pointer), amelyek a mindenkori legkisebb helyiértékű BCD byte-ot megelőző byte-ra mutatnak. A két érték összeadására alkalmas rutin törzse ilyen lehet:

```
hossz                EQU       5
                     •
                     •
össz – BCD – törzs:  MOVEA.W   # FORRÁSPTR,A0
                     MOVEA.W   CELPTR,A1
                     MOVE.W    # hossz –1,D0
összciklus:          ABCD      –(A0), –(A1)
                     DBF       D0, összciklus
                     •
```

Először a két mutató (pointer) az adatok előtti byte-okra állítódik, egy adatregiszterben egy szót egy számláló érték inicializál, és aztán a DB_{CC} ciklusban megtörténik a tulajdonképpeni összeadás. Ennek a ciklusnak a nyilvánvaló egyszerűsége mellett (itt a feltétel mindig hamis, a ciklus tehát csak akkor fejeződik be, amikor a D0 adatregiszterben az alsó szó –1 lesz), még az az előnye, hogy a bizonyos mértékben cache-orientált 68010-es processzorral és az ezt követő processzorokkal rendkívül gyorsan fel lehet dolgozni, mivel ez a ciklus ebben az esetben a cache feltételeknek teljes mértékben megfelel.

Ezek a feltételek kikötik, hogy az esetleges visszaugrás távolsága nem lehet 4 byte-nál nagyobb, és hogy az utasítás csak a ciklust (LOOP) végrehajtani képes címzésmóddal használható. Ezeket Az utasításkészlet című fejezet bevezetőjében soroltuk fel.

A cikluson belül a 68000-es először dekrementálja a két címregisztert, hogy aztán hozzáférhessen az első byte-hoz. Ez az első menetben a tárban a legmagasabb címen levő, tehát a legalacsonyabb helyiértékű byte. Ez a két byte összeadódik, és a célregiszterbe kerül. Ezután a processzor megvizsgálja a ciklus DBF részét arra vonatkozóan, hogy a megadott feltétel teljesül-e.

Mivel az F (= false, hamis) soha nem lehet igaz, a ciklus itt nem fejeződik be.

Ezután a megadott D0 adatregiszter alsó szava dekrementálódik, itt tehát 4-ről 3-ra csökken. Miután ez az érték nem –1, ugrás következik a megadott címre, tehát a következő összeadáshoz. Ez mindaddig ismétlődik, míg az adatregiszter a –1 értéket eléri, jelen esetben tehát 5 menet után. Ennek során mind az öt BCD byte összeadódott, az eredmény a célként megadott regiszterben van, és a feldolgozás befejeződött.

A programozói nyelvek támogatásáról szóló fejezetben szó volt a LINK és az UNLK alkalmazásáról, és említettük, hogy ezekkel a műveletekkel könnyen készíthetők rekurzív algoritmusok.

A feldolgozási folyamatok egyik speciális csoportját nevezik rekurzív folyamatnak. Rekurzívak azok a folyamatok, amelyek képesek arra, hogy saját maguk lássák el önmagukat új bemeneti értékekkel. Ilyen rutinok írása hétköznapi dolog, de a szokásos iterációval gyakran olyan algoritmusokat kapunk, amelyek ugyan működőképesek, de túlságosan körülményesek. Például egy labirintusban a kijárat vagy valamilyen más célpont megtalálásához valamennyi lehetséges utat meg kell vizsgálni, és az elsőként megtalált kivezető út jelent egy megoldást.

Az algoritmus azonban úgy is megírható, hogy az a labirintus útjain lépésről lépésre „rágja át" magát, és az egyszer már használhatatlannak bizonyult úton még egyszer nem halad át. Ilymódon lényegesen gyorsabb és érdekesebb algoritmusokat kapunk.

A rekurzív folyamatok alkalmazását a faktoriálisszámítás példáján mutatjuk be. Ezt a matematikai műveletet, amelynek jelölésére a ! jel használatos, a diákoknak így tanítják:

$n! := 1*2*3*\ldots*n$ és $0! := 1$

Ez a művelet azonban más képlettel is megadható:

$n! := n*(n-1)!$ és $0! := 1$

Ha például az $n=3$, akkor a megadott képlettel így számítjuk ki a faktoriálist:

$3! := 3*(3-1)! = 3*2!$
$2! := 2*(2-1)! = 2*1!$
$1! := 1*(1-1)! = 1*0!$
$0! := 1$

itt befejeződik a rekurzió, mivel egy olyan, konkréten definiált értékbe ütközik, ami tovább már nem számítható.

Mivel ezt az utolsó értéket ismerjük, a fölötte álló három egyenlőség most alulról felfelé felgöngyölíthető:

$1! := 1*0! = 1*1 = 1$
$2! := 2*1! = 2*1 = 2$
$3! := 3*2! = 3*2 = 6$

Itt befejeződik a faktoriálisszámítás, mert egyértelmű eredményt kaptunk.

Ez a számítási mód az első ránézésre nyilván sokkal bonyolultabbnak tűnik ahhoz képest, mintha a számítást részekre bontanánk, és az egyes szorzatokat egy cikluson belül számítanánk ki. Az előző módszer azonban a programot áttekinthetőbbé teszi, mert így eléggé egyszerűen biztosítható, hogy az algoritmus mindig működjön, ill. a hibahelyzeteket felismerje. Általában a matematikában is könnyebben bizonyítható az ilyen, rekurzív módon definiált kifejezések helyessége, mint az egymást követő elemek szorzatai.

Az itt alkalmazott eljárásnál abból indulunk ki, hogy az esetleges paraméterek mindig kettős-szó hosszúságban helyezkednek el a veremben, mégpedig az első paraméter a legalsó tárcímen. A mindenkori folyamatban a D0, D1 és az A0 regisztereken kívül valamennyi regiszter értéke tárolásra kerül.

Egy függvény (művelet) értékének a visszaadása mindig a D0 regiszterben történik; ha egynél több értéket kell visszaadni, akkor erre a célra a D1 és az A0 regiszter szolgál. Ha még ezek sem lennének elegendők, akkor a regiszterek egyikében azt a címet kell átadni, amelyik egy értékvisszaadó táblázat elejére mutat.

```
LB                   EQU         A6
visszatérési érték   EQU         D0
dec.reg.             EQU         D1
kozbossz             EQU         D2
param 1              EQU         8
faktoriális:         LINK        LB, #0
; az egyetlen használt, tárolandó kozbossz re-
; giszter tárolása és a paraméter átvétele
                     MOVE.L      kozbossz, – (SP)
                     MOVE.L      param 1 (LB), kozbossz
; a 0 rekurziós végérték vizsgálata
                     CMP.W       kozbossz, #0
                     BLE         kesz
; itt még szükség van egy rekurziós lépcsőre,
; a −1 paraméter a verembe kerül
                     MOVE.W      kozbossz, dec.reg.
                     DEC.W       dec.reg.
                     MOVE.L      dec.reg, – (SP)
                     CALL        faktoriális

; ezen a helyen a rekurziós lépcső valamennyi
; eleme visszatér
                     MULU        kozbossz, visszatérési érték
faktret:             MOVE.L      (SP)+, kozbossz
                     UNLK        LB
                     RET
; a faktoriális (0) visszaadási érték 1
kesz:                MOVE.L      #1, visszatérési érték
                     JMP         faktret
```

Ez a program több részre van felbontva. Az első rész a paramétereket veszi át, és az itt biztonságba helyezendő egyetlen regiszter tárolását végzi. Az A6 regiszter a lokális bázis, az A7 pedig a normál veremmutató (SP).

Ha most meghívjuk a „faktoriális" függvényt, akkor először a következő veremképet kapjuk:

paraméter
visszatérési cím
SP LB → az A6 értéke

Az átadott paraméter értéke most a „kozbossz" regiszterbe kerül, és a program második része azt vizsgálja meg, hogy az értéke elérte-e a nullát. Ha nem, akkor a faktoriális ismételten meghívásra kerül, de előtte a kozbossz–1 érték a verembe töltődik. Ha ez a meghívás mégegyszer visszatér, akkor a „faktoriális (kozbossz–1)"-ből a D0-ba visszaadott eredmény és a kozbossz összeadásra kerül, és betöltődik a D0-ba. Ez a folyamat tehát eleget tesz a követelményeknek.
Ha viszont a kozbossz értéke nulla, vagy kettes komplemensben negatív, akkor a visszadásra kerülő érték 1.
Itt érdemes megfigyelni, hogyan néz ki a verem a faktoriális különböző meghívásai esetén. Legyen az első meghívó paraméter értéke 3:

3
visszatérési cím 1
SP LB1 → az A6 értéke

Itt mégegyszer bemutatjuk a kiindulási helyzetet; a kozbossz-nak van valamilyen értéke, amit tárolni kell. Az első rekurzív meghívás után a verem a következő módon nő:

3
visszatérési cím 1
LB1 → az A6 értéke
a D2 értéke
2 = új paraméter
visszatérési cím 2
SP LB → LB1

A faktoriális művelet így paraméterként a 2-es értéket kapja, mivel ez az éppen aktuális bázishoz képest mindig relatív.

A kozbossz értéke itt 3, amelyet elsőként kell a verembe helyezni, azt megelőzően, hogy a faktoriálist újra meghívnánk:

3
visszatérési cím 1
LB1 → az A6 értéke
a D2 értéke
2
visszatérési cím 2
LB2 → LB1
3 = kozbossz
1 = új paraméter
visszatérési cím 3
SP LB → LB2

A feldolgozás ugyanúgy történik, mint az előző esetben. Az „1" paraméter betöltődik a veremből a kozbossz-ba, miután ez a regiszter 2-es értékkel a verembe kerül. Mivel ez az „1" nem egyenlő „0"-val, a faktoriális ismét meghívásra kerül, miközben a kozbossz értéke 1 lesz.

3
visszatérési cím 1
LB1 → az A6 értéke
a D2 értéke
2
visszatérési cím 2
LB2 → LB1
3 = kozbossz
1
visszatérési cím 3
LB3 → LB2
2 = kozbossz
0 = új paraméter
visszatérési cím 4
SP LB → LB3

Itt a művelet megállapítja, hogy a paraméter nulla, és a D0 regiszterbe az utolsó felhívási helyre egy „1" kerül vissza. A CALL faktoriális mögött rögtön megtörténik a visszaadott érték és az ezen a helyen levő kozbossz összeszorzása; ami itt tehát 1*1. Ez az „1" visszatérési érték az 1*0! művelet eredménye, és visszakerül a rekurzió következő lépcsőjére. Ez a rekurziós lépcső most ismét a korábbi CALL faktoriálisra lép, és megszorozza az itt megjelenő, „1" visszatérési értéket a kozbossz-szal, aminek ez értéke a veremben levő, megelőző lépcső ismétlése után 2 lesz. Ez a 2*1 szorzat a 2*1! kiszámítását jelenti, és az eredménye 2. Ez a visszaadott érték a D0 regiszterben ismét a fölötte levő rekurziós lépcsőre kerül, ami ezt az értéket megszorozza a kozbossz-szel. Az ebben a lépcsőben a veremből elővett kozbossz értéke 3, így tehát a 3*2 művelet kerül kiszámításra, ami megegyezik a 3*2! művelettel. A művelet eredménye (6) visszakerül a felhívó szintre, ami ebben az esetben a faktoriális első meghívása volt.

A veremmutató minden egyes UNLK-RET ciklusban inicializálásra kerül, úgy, hogy ezen a helyen mindig az az érték van, ami a művelet első meghívása előtt volt.

Ez a kis példa azt mutatja be, hogy az ilyen veremorientált művelet a LINK-UNLK utasításokkal milyen nagy teljesítményekre képes.

A processzor alkalmazója egy (helyes!) művelet vagy feldolgozási folyamat meghívásakor megbízhat abban, hogy egy regiszter csak pontosan ellenőrzött módon változtatja meg állapotát.

A korábbi generációs, 8 bites operációs rendszer használójától gyakran hallott „ajajj, ebbe a regiszterbe még ez is kellene” megjegyzésre most már nincs szükség.

FÜGGELÉK

Felhasznált irodalom

COFFMAN; ELPHICK; SHOSHANI: System Deadlocks. Computing Surveys, vol. 3, S. 68–78, Juni 1971

COFFMAN; DENNING: Operating Systems Theory. Englewood Cliffs N.Y,; Prentice Hall, 1973

DIJKSTRA, E. W.: A short introduction into the art of programming. TU Eindhoven, Niederlande, August 1971

DIJKSTRA, E. W.: Solution of a Problem in Concurrent Programming. CACM, vol. 8, S. 569, September 1965

DIJKSTRA, E. W.: Co-operating Sequential Process, in Programming Languages. F. Genuys, N.Y.: Academic Press, 1968

HABERMANN, A. N.: Prevention of System Deadlocks. CACM, vol. 12, S. 373–377, Juli 1969

HILF, WERNER: Neue Peripheriebausteine für den 68000, Elektronik 19/1981

KANE; HAWKINS; LEVENTHAL: 68000 Assembly Language Programming. Verlag Osborne/McGraw Hill, 1. Auflage 1981

MOSTEK CORP.: Microelectronic Data Book. May 1982

MOTOROLA INC.: Interfacing 6800-Peripherals to the MC68000 Asynchronously. Application Note An808

MOTOROLA INC.: MC68451 Memory Management Unit (MMU). Datenblatt

MOTOROLA INC.: Interrupt Sychronisation on the MC68000 (Maske T 6E). Engineering Bulletin EB84

MOTOROLA INC.: A Discussion of Interrupts for the MC68000. Engineering Bulletin EB97

POL, BERND: Betriebssysteme – Eine Einführung. Elektronik 2/1981

ROCKWELL INTERNATIONAL CORP.: R 684 65 Double-Density Floppy Disk Controller. Datenblatt

ROCKWELL INTERNATIONAL CORP.: R68560, R68561 MPCC. Datenblatt

SCANLON, LEO J.: The 68000: Principles and Programming. Howard W. sams Co, Indianapolis

SIGNETICS CORP.: SCB68430 Direct Memory Access Interface (DMAI). Datenblatt

STUHLMÜLLER, PETER: Aufgaben eines Betriebssystems. Elektronik 2/1981

TANENBAUM, ANDREW S.: Structured Computer Organisation. Prentice/Hall International editions, 1976

WIRTH, NIKLAUS: The programming language Pascal. Acta Informatica, S. 35–63, 1971

WIRTH, NIKLAUS: On Multiprogramming, Machine Coding and Computer Organization. CACM, vol. 12, S. 489–498, September 1969

Védett nevek

MC68xxx	Motorola Inc.
PDP–11	Digital Equipment Corp. (DEC)
LSI–11	Digital Equipment Corp. (DEC)
Z8000	Zilog
NSC16032	National Corp.
TMS9900	Texas Instruments
iAPX86	Intel Corp.
CP/M	Digital Research
UNIX	Bell Laboratories, AT & T
Mickey Mouse	Walt Disney Corp.

TÁRGYMUTATÓ

68008 79
68010 83
68012 95
68020 96

A

Abszolút címzés, kettős-szó hossz 150
Abszolút címzés, szó hossz 149
Adatbusz 43
Adatok címzése 68020-asnál 102
Adatok tárbeli szervezése 22
Adatregiszter közvetett 142
Adatregiszter közvetlen 138
Állapotregiszter 19
Aritmetika társprocesszor 130
Aszinkron buszhozzáférés 52
Aszinkron buszvezérlés 43
Aszinkron írási ciklus 53
Aszinkron olvasási ciklus 52
ADATOK kategória (címzésmód) 153

B

Bus Arbitration Modul 126
Buszciklus lezárása 75
Buszkezelő egység (BAM) 126
Buszmegszakító modul (BIM) 122
Buszműveletek (68020) 111
Buszvonal átadás vezérlés 45
Buszvonal kérése 69
BAM (buszkezelő egység) 126
BCD összeadás 347
BIM (buszmegszakító) 122

C

Cache-tár 68020-asnál 104
Ciklus (Loop) mód 93, 157
Címbusz 43
Címregiszter közvetlen 138
Címzésmódok 137
CPU-space 68020-asnál 113

D

DMA (közvetlen tárhozzáférés) 124

E

Effektív cím 137
Exceptions (kizárások) 25
ELLENŐRZŐ kategória (címzésmód) 153
EPCI 129

F

Fejlesztés 11
Felépítés (architektúra) 13
Feltételkódok 156
Flags (kapcsolók) 155
Funkciókód-kimenetek 46
Funkciókódok 24
FPP 132

H

Hozzáférési szabályok 23

I

Interrupts (megszakítások) 66
IPC (Intell. Perifériaell.) 121

J

Jelek a 68000-esnél 14
Jelek a 68020-asnál 106

K

Kapcsolók (Flags) 155
Kizárás folyamatábrája 34
Kizárások (Exceptions) 25
Kizárások a 68020-asnál 116
Kizárások feldolgozása 25
Kizárások prioritásai 31
Közvetlen tárhozzáférés (DMA) 124
Külső nVMA-generálás 80
Külső/belső jelek kapcsolata 48

L

Lábkiosztás, 68008 82
Lábkiosztás, 68020 106
Lábkiosztás, 68000 Dual-In-Line 49
Lábkiosztás, 68000 PIN-GRID 50
Loop (ciklus) mód 93, 157

M

Megszakítás vezérlése 45
Megszakítások (Interrupts) 66
Megszakító modul (BIM) 122
Mikroprogramozás 13
Modultechnika 343
MFP (Multi-Function-Peripheral) 130
MMU (Tárkezelő egység) 127, 332
MPCC 129

O

Olvasás-módosítás-írás ciklus 58
Operációs rendszerek 329
Operációs rendszerek támogatása 336
Operandus hossza 136
Operandusok formátuma 18

P

Paging 332
Paraméterátadás 340
Párhuzamos I/0 123
Programnyelvek támogatása 340
Programozási modell (68000-es) 17
Programozási modell (68010-es) 86
Programozási modell (68020-as) 98
Programozási példák 347
Programszámláló relatív 144
PC indirekt előzetes indexszel 148
PC indirekt 32 bites indexszel 147
PC indirekt utólagos indexszel 147
PC indirekt indexszel 145
PC relatív 144
PC relatív adatregiszter indirekt 147
PI/T 123

R

Regiszter direkt 138
Regiszter indirekt 138
Regiszter indirekt, indexszel 141
Regiszter indirekt címkülönbséggel és index 141
Regiszter indirekt címkülönbséggel 140
Regiszter indirekt előzetes dekrementálás 139
Regiszter indirekt utólagos inkrementálás 139
Regiszterkészlet (68010) 84
Regiszterszerkezet 16

Regiszterszerkezet (68020) 97
Regiszterszervezés 19
Rekurzió 348
Rendszer órajel 42
Rendszervezérlés 46, 71
RTE utasítás 68010-esnél 92

S

Soros I/0 129
Szegmentálás 332
Szemaforok 337
Szinkron buszhozzáférés 61
Szinkron buszvezérlés 44

T

Tápfeszültség 42
Tár indirekt előzetes indexszel 143
Tár indirekt utólagos indexszel 142
Tárkezelő egység (MMU) 127, 332
Társprocesszor interfész 118
Társprocesszor, aritmetika 130
Társzervezés 68020-asnál 101
Technológia 13
Töréspontok a 68020-asnál 114
TÁR kategória (címzésmód) 153

U

Újrafutási (Re-run) ciklus 73

Ü

Üzemi állapotok 23

V

Vektortáblázat 26
Veremtartalom kizárásoknál 32
VÁLTOZTATHATÓ kategória (címzésmód) 153

Táblázatok

Disassembler táblázat

A 68000-es sorozat processzorainak rendkívül összetett felépítése miatt ez a táblázat gyakran csak utalásokat képes adni a disassemblálandó műveleti kódokra. Így ha egy kód a táblázat szerint egyértelműen nem disassemblálható, akkor erre vonatkozóan a táblázatban szereplő mnemonikok egyikét meg kell vizsgálni.

$00	CMP2, CHK2, MOVEP, ORI
$01	BSET, BTST BCHG, BCLR
$02	ANDI
$04	SUBI
$06	ADDI, RTM CALLM
$08	CAS, CAS2
$0a	EORI
$0c	CMPI
$0e	MOVES
$1x	MOVE.B
$2x	MOVE.W
$3x	MOVE.L
$40	NEGX, MOVE f. SR
$41	CHK, LEA
$42	CLR, MOVE f. CCR
$44	NEG, MOVE to CCR
$46	NOT, MOVE to SR
$48	MOVEM, EXT, NBCD, PEA, SWAP, TRAP, BKPT
$49	EXTB
$4a	TST, TAS ILLEGAL
$4c	DIVSL, DIVUL
$4e	MOVE, USP, JMP, JSR, MOVEC, NOP, LINK, RTD, RESET, RTE TRAPV, RTR, UNLK, STOP, RTS
$50	ADDQ, DBcc Scc, TRAPcc
$51	SUBQ, SUBX
$60	BCC, BRA, BSR
$7x	MOVEQ
$80	DIVU, OR
$81	DIVS, PACK, SBCD, UNPK
$90	SUB, SUBA
$ax	foglalt
$b0	CMP, CMPA, EOR
$b1	CMPM
$c0	MULU, AND
$c1	ABCD, EXG, MULS
$d0	ADD
$d1	ADDX
$e0	ROL, ROR, ROXL, ROXR, ASL, ASR, MOVEQ
$e1	LSL, LSR
$e8	BFTST
$e9	BFEXTU
$ea	BFCHG
$eb	BFEXTS
$ec	BFCLR
$ed	BFFF0
$ee	BFSET
$ef	BFINS
$fx	társprocesszor

Sorsz.	Címzésmód	Módus	Reg.	ADATOK	TÁR	ELLEN-ŐRZÉS	VÁLTOZ-TATHATÓ	Szintaxis	Csak 68020
1	Adatregiszter közvetlen	000	Nr.	x	–	–	–	Dn	
2	Címregiszter közvetlen	001	Nr.	–	–	–	x	An	
3	Címregiszter közvetett	010	Nr.	x	x	x	x	(An)	
4	Címregiszter közvetett utólagos inkrementálással	011	Nr.	x	x	–	x	(An) +	
5	Címregiszter közvetett előzetes dekrementálással	100	Nr.	x	x	–	x	– (An)	
6	Címregiszter közvetett, címkülönbséggel	101	Nr.	x	x	x	x	d_{16}(An)	
7	Címregiszter közvetett, 8 bites címkülönbséggel	110	Nr.	x	x	x	x	d_8(An, Rn)	
8	Címregiszter közvetett, indexszel	110	Nr.	x	x	x	x	bd (An, Rn)	*
9	Tár közvetett, előzetes indexszel	110	Nr.	x	x	x	x	(bd, An, Rn, od)	*
10	Tár közvetett, utólagos indexszel	110	Nr.	x	x	x	x	(bd, An, Rn, od)	*
11	Abszolút szóhosszúság	111	000	x	x	x	x	xxxx. W	
12	Abszolút kettősszó-hosszúság	111	001	x	x	x	x	xxxxxxxx. L	
13	Közvetlen (adatok)	111	100	x	x	–	–	#„adatok”	
14	Programszámláló közvetett, 16 bites címkülönbséggel	111	101	x	x	x	–	d_{16}(PC)	
15	Programszámláló közvetett, 8 bites indexszel	111	011	x	x	x	–	d_8(PC, Rn)	
16	Programszámláló közvetett, indexszel	111	011	x	x	x	–	bd (Pc, Rn)	
17	Programszámláló közvetett, előzetes indexszel	111	011	x	x	x	–	(bd, PC, Rn, od)	
18	Programszámláló közvetett, utólagos indexszel	111	011	x	x	x	–	(bd, PC, Rn, od)	

A címzésmódok számozása a következő oldalon lévő táblázatra is vonatkozik.

Mnemonik	Feltétel	Kód Bitkombináció
T	True	0000 mindig
F	False	0001 soha
HI	High	0010/C*/Z
LS	Low or Same	0011 C+Z
CC/HS	Carry Clear Higher or Same	0100/C
CS/LO	Carry Set/Lower	0101 C
NE	Not Equal	0110/Z
EQ	EQual	0111 Z
VC	Overflow Clear	1000/V
VS	Overflow Set	1001 V
PL	PLus	1010/N
MI	MInus	1011 N
GE	Greater or Equal	1100 N*v+/N*/V
LT	Less Than	1101 N*/V+/N*V
GT	Greater Than	1110 N*V*/Z+/N*/V*/Z
LE	Less or Equal	1111 Z+N*/V+/N*V

* logikai AND (ÉS)

+ = logikai OR (VAGY)

/ = logikai NOT (NEM)

Ára: 349,–Ft

SZÁMÍTÁSTECHNIKA A KÖNYVESBOLTOKBAN

2C

ÜZLETHÁLÓZAT

NOVOTRADE – 2C ÁRUHÁZ

1136 Bp., Balzac u. 35. Tel.: 402-954

Az alább felsorolt üzletekben már az Önök rendelkezésére állunk:

ÁLLAMI KÖNYVTERJESZTŐ V. – NOVOTRADE 2C

BUDAPEST

Táncsics Könyvesbolt
1073 Lenin krt. 17.
Telefon: 422-178

BUDAPEST

Műszaki Könyváruház
1061 Liszt Ferenc tér 9.
Telefon: 420-353

MŰVELT NÉP KÖNYVTERJESZTŐ V. – NOVOTRADE 2C

PÉCS

Zrínyi Miklós Könyvesbolt
7621 Jókai u. 25.
Telefon: 72-12835

VESZPRÉM

Kölcsey Ferenc Könyvesbolt
8200 Cserhát u. 7.

SZEGED

Tömörkény Könyvesbolt
6720 Lenin krt. 48.
Telefon: 62-21453

DEBRECEN

Szak- és ismeretterjesztő
Könyváruház
4024 Hunyadi u. 8.
Telefon: 52-23237

BÉKÉSCSABA

Radnóti M. könyvesbolt
5600 Tanácsköztársaság út 2.
Telefon: 25-207

SZOLNOK

Szigligeti Könyvesbolt
5000 Ságvári krt. 35.
Telefon: 56-11133

SZOMBATHELY

Savaria Könyvesbolt
9700 Mártírok tere 1.
Telefon: 94-12341

GYŐR

Pattantyús Á. Géza Szakkönyvesbolt
9021 Molnár Ferenc u. 9.

MISKOLC

Chip-kuckó
3530 Tanácsház tér 14.

Minden érdeklődőt szeretettel vár
az ÁKV, a Művelt Nép és a NOVOTRADE RT.!